

# **TWITTER-CONTROLLED MICROCONTROLLER SYSTEM FOR HOME AUTOMATION**

By

MUHAMMAD GADDAFI BIN RUSLI

Dissertation submitted in partial fulfillment  
of the requirements of the  
Bachelor of Engineering (Hons)  
(Electrical & Electronics Engineering)

DECEMBER 2010

Universiti Teknologi Petronas  
Bandar Seri Iskandar  
31750 Tronoh  
Perak Darul Ridzuan

# **CERTIFICATION OF APPROVAL**

## **TWITTER-CONTROLLED MICROCONTROLLER SYSTEM FOR HOME AUTOMATION**

by

Muhammad Gaddafi bin Rusli

A project dissertation submitted to the  
Electrical & Electronics Engineering Programme  
Universiti Teknologi PETRONAS  
in partial fulfilment of the requirement for the  
Bachelor of Engineering (Hons)  
(Electrical & Electronics Engineering)

Approved:

---

Dr Mohd Zuki bin Yusoff  
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

December 2010

## **CERTIFICATION OF ORIGINALITY**

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

---

Muhammad Gaddafi bin Rusli

## **ACKNOWLEDGEMENT**

Given this opportunity, I would like to acknowledge those who have been helping me, either directly or indirectly, in continue working on the project – starting from exploration, to the development phase and until the completion phase of this Final Year Project in Universiti Teknologi PETRONAS.

First of all, I would like to express my gratitude to God for giving me the strength and health throughout my years of study in this university, that allows me to gain enough theoretical knowledge and practical skill sets as an electronics engineer, to be applied into this particular project or other project in the coming years.

I would like to thank Dr Mohd Zuki bin Yusof, my supervisor for this project for the past two semesters. Dr Zuki has been giving me moral and mental support since the first day I proposed the idea of this project to him, in early 2010. His understanding and confidence on my capabilities and passion towards the idea has given me the flexibility and spiritual advantages that I needed in conducting the project from start to finish.

Besides, I also like to thank all the lab technicians in the Electrical and Electronics Engineering department for their assistants in helping me to obtain some of the tools and resources needed during the development stage. The same gratitude goes to my colleagues in the university for the support and interest in this small project. Your interest in this project has boosted my confidence and expands my capabilities in exploring further into the project.



## **RECOGNITIONS**

As of the time of this writing, Twitter-controlled microcontroller System for Home Automation project has obtained the following achievements:

- (1) Grade 4.00 in Final Year Project 1 – among top projects in Electrical and Electronic Engineering department (June 2010)
- (2) Chosen among the Top 10 project for Electrex (October 2010)
- (3) Received Silver Medal in EE Category during the 26<sup>th</sup> edition of Engineering Design Exhibition (October 2010)

## **ABSTRACT**

Twitter-controlled microcontroller system for home automation is a system that can be used to control certain part of a residential house via the Internet or mobile devices. It aims to illustrate the ability of a web application to be integrated in a real world application – which is represented by home automation in this particular project. This project is significance in a sense that it can improve the lifestyle of the targeted group in having an alternative control over their premise without the physical presence in that particular premise. It is implemented by making use of the existing platform available in the Twitter infrastructure and wide audience, as a unique means of transmitting information and instruction. This instruction is then fetched by the system and is propagated through the system via several processes and flows. The project involves software development on host computer, hardware interfacing between host computer and custom device, wireless communication and transmission; and the end part on the devices automation. The final outcome of this project is a fully working prototype that can demonstrate the home automation capabilities via several mediums. It is hoped that this project will benefit various kind of users especially those who have a busy lifestyle and have the need to control their house when they are on the move, or anyone who would like to have an alternative ways of automating some devices in their houses.

## **TABLE OF CONTENTS**

<b>CERTIFICATION OF APPROVAL .....</b>	<b>ii</b>
<b>CERTIFICATION OF ORIGINALITY .....</b>	<b>iii</b>
<b>ACKNOWLEDGEMENT .....</b>	<b>iv</b>
<b>RECOGNITIONS .....</b>	<b>v</b>
<b>ABSTRACT .....</b>	<b>vi</b>
<b>TABLE OF CONTENTS .....</b>	<b>vii</b>
<b>LIST OF TABLES .....</b>	<b>xi</b>
<b>LIST OF FIGURES .....</b>	<b>xii</b>
<b>ABBREVIATIONS AND NOMENCLATURES .....</b>	<b>xiii</b>
<b>CHAPTER 1: INTRODUCTION .....</b>	<b>1</b>
1.1    Background .....	1
1.2    Problem Statement .....	2
1.3    Project Objectives .....	3
1.4    Scope of study .....	3
1.5    Report Organization .....	4
<b>CHAPTER 2: LITERATURE REVIEW .....</b>	<b>5</b>
2.1    Twitter Application Programming Interface .....	5
2.2    Microcontroller Programming .....	6
2.3    Real-life Twitter Applications .....	7

2.4	USB-based Devices .....	8
2.5	Home Automation Technologies .....	8
<b>CHAPTER 3: METHODOLOGY .....</b>		<b>9</b>
3.1	Project Development Components .....	9
3.2	Project Development Stages .....	11
3.3	Web-based Interface .....	13
3.3.1	<i>Introduction</i> .....	13
3.3.2	<i>Twitter API and OAuth authentication</i> .....	14
3.3.3	<i>Overcoming duplication limitation</i> .....	17
3.4	Status Fetcher and XML Parser .....	18
3.4.1	<i>Introduction</i> .....	18
3.4.2	<i>XML Parser</i> .....	18
3.5	Custom Windows Application .....	20
3.5.1	<i>Introduction</i> .....	20
3.5.2	<i>USB connectivity</i> .....	20
3.5.3	<i>Actions and interval definitions</i> .....	21
3.5.4	<i>Periodic content fetch from text file</i> .....	22
3.5.5	<i>User interface development</i> .....	24
3.6	USB Device Development: Hardware .....	25
3.6.1	<i>Introduction</i> .....	25
3.6.2	<i>Transfer types</i> .....	26
3.6.3	<i>Microcontroller circuits</i> .....	27

3.6.4	<i>Converting to Printed Circuit Board</i> .....	30
3.7	USB Device Development: Firmware .....	30
3.7.1	<i>Introduction</i> .....	31
3.7.2	<i>Microchip Application Library – USB framework</i> .....	31
3.7.3	<i>Customizing the firmware</i> .....	32
3.7.4	<i>I/O configuration</i> .....	32
3.8	Wireless Transmission .....	35
3.8.1	<i>Introduction to Xbee wireless module</i> .....	35
3.8.2	<i>Configuration setting</i> .....	36
3.8.3	<i>Wireless module circuits</i> .....	37
3.8.4	<i>Transmitter and receiver</i> .....	32
3.9	Small-scale Home Devices Model .....	41
3.9.1	<i>Introduction</i> .....	35
3.9.2	<i>Microcontroller circuits</i> .....	36
3.9.3	<i>House model</i> .....	37
<b>CHAPTER 4: RESULTS AND DISCUSSION</b> .....		<b>43</b>
4.1	Full System Prototype .....	43
4.2	Usage Requirements .....	46
4.3	Full System Flow and Integration .....	47
4.4	USB Device Failure Analysis .....	49
4.5	Optimal Fetch and Feed Intervals .....	51

<b>CHAPTER 5: CONCLUSION AND RECOMMENDATIONS .....</b>	<b>52</b>
5.1 Conclusion .....	52
5.2 Recommendations .....	53
 <b>REFERENCES .....</b>	 <b>54</b>
APPENDIX A: Source Code for Web-based Interface Application .....	55
APPENDIX B: Screenshots of the Web-based Application .....	63
APPENDIX C: XML-formatted Status Information .....	66
APPENDIX D: Source Code for Status Fetcher and XML Parser .....	67
APPENDIX E: Source Code for Custom Windows Application .....	68
APPENDIX F: USB Device Schematic .....	78
APPENDIX G: USB Connectors and Ports .....	79
APPENDIX H: Source Code for USB Device Firmware .....	80
APPENDIX I: XBee Breakout Boards .....	88
APPENDIX J: XBee Wireless Module Circuit Schematic .....	89
APPENDIX K: Source Code for Wireless Transmission .....	90
APPENDIX L: Source Code for Home Devices .....	94
APPENDIX M: Project Gantt Charts .....	100

## LIST OF TABLES

Table 1 : Scope of study for the project .....	3
Table 2 : Programming languages to be used for the web-based interface .....	13
Table 3 : Comparison between USB, Serial and Parallel port .....	25
Table 4 : USB transfer types .....	26
Table 5 : Pins for Type A USB connector .....	30
Table 6 : Actions definitions and representations .....	34
Table 7 : Basic specifications of XBee Wireless Module .....	35
Table 8 : Four basic pin assignments on XBee Wireless Module .....	37
Table 9 : Designated character for each transmitter input .....	40
Table 10 : Small-scale home device automation representations .....	41
Table 11 : Usage requirements of the system .....	46
Table 12 : Stress test with 1 second timer interval .....	50
Table 13 : Stress test with 10 second timer interval .....	50
Table 14 : Stress test with 30 second timer interval .....	50

## LIST OF FIGURES

Figure 1 : Four major components involved in the project .....	9
Figure 2 : Overall project development stages .....	12
Figure 3 : Setting page for the custom application registered with Twitter .....	15
Figure 4 : Twitter user will be prompted for external access .....	15
Figure 5 : XML-formatted information from a single status update .....	18
Figure 6 : Visual Studio development environment .....	24
Figure 7 : Pin-out of PIC18F4550 .....	27
Figure 8 : Built-in components in PIC 18F4550 .....	28
Figure 9 : Simple circuit schematic for USB device with 18F4550 .....	29
Figure 10 : Four types of USB connector .....	29
Figure 11 : Configuring XBee using X-CTU .....	36
Figure 12 : Illustration of wireless transmission configuration .....	37
Figure 13 : Basic four pins connection from XBee module to microcontroller .....	38
Figure 14 : Point-to-point transmission demonstration from A(TX) to B(RX) .....	40
Figure 15 : Screenshot of the custom Windows application .....	34
Figure 16 : USB device constructed on breadboard .....	34
Figure 17 : USB device constructed on custom PCB .....	34
Figure 18 : Small-scale model of the house .....	35
Figure 19 : Receiver modules and other home devices circuits .....	35
Figure 20: Accessing Twitter using mobile application .....	47
Figure 21: Accessing Twitter using desktop application .....	47



## **ABBREVIATIONS AND NOMENCLATURES**

API	Application Programming Interface
CDC	Communications Device Class
CSS	Cascading Style Sheet
DC	Direct Current
DIY	Do-it-yourself
GUI	Graphical User Interface
HID	Human Interface Device
IEEE	Institute of Electrical and Electronics Engineers
JSON	JavaScript Object Notation
LCD	Liquid Crystal Display
LED	Light Emitting Diode
MSD	Mass Storage Device
MAL	Microchip Application Library
PHP	Hypertext Processor
PCB	Printed Circuit Board
RF	Radio Frequency
URL	Universal Resource Locator
USB	Universal Serial Bus
xHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language

# **CHAPTER 1**

## **INTRODUCTION**

Twitter-controlled microcontroller system for home automation is a project that aims to illustrate the ability to integrate Internet application, in this case, Twitter, to be connected to a real world application. For this particular project, home automation system is chosen to illustrate the end product for a real-world application

### **1.1 Background**

Twitter, the Internet application chosen for this project, is a real-time status update application developed by Obvious Inc based in United States. It is one of the fastest rising social networking services on the Internet, with millions of users from all over the world and more than 100 million status updates sent out per day. Their user ranges from individuals (including various public figures) to huge respectable companies (such as Google).

Twitter is chosen for the project because of its huge user base in many countries, including Malaysia. Besides, it is a portable and versatile platform for a remote system, as it can be access either via their website, various desktop applications for all operating system, mobile applications or even via short messaging services (SMS). It has a flexible API (Application Programming Interface), which is used to develop various types of applications and services by independent and third party developers. In this project, a web-based application will also be developed to serve as a simple control panel for a much friendlier approach for the targeted users. Looking into this, user can easily set up a free Twitter account and start controlling their devices from their computer or mobile phone, at anytime and anywhere in the world.

The basic model of the application is: user will send a textual message in the range of 140 characters as *status update*, anytime they wanted to. This status update will appear on their *timeline* in real time. This model will be used in the project, where those status updates will be treated as the string of instructions, sent by the user, to the microcontroller system. Then, these instructions will be translated into pre-determined actions that can be used to control pre-determined devices around the house. Compared to other existing home automation technology, this system aims for a much simpler setup and more of a DIY basis, thus reducing the installation and maintenance cost in the long run.

## **1.2 Problem Statements**

This project, or rather the end product of the project, can be used by anyone as an alternative for a home automation system. However, it does have a specific target group. For people who are always go out for business travel or vacation, it is a huge concern to leave their houses unattended for days, especially if they lived on their own. Besides, these targeted people normally come from a group of society with busy lifestyle and tight schedules.

Therefore, it is much easier to have an integrated, portable and simple system that can be used to control certain parts of their house even if they were away. The project aims to provide an alternatively easy way for targeted users to manage and control their household devices and applications, via several mediums (Internet and mobile access), especially when they were not around the house.

### 1.3 Project Objectives

The main objective of this project is to develop an alternative method for home automation system, paired with an Internet service called Twitter as the medium for control mechanism. Other than that, another main objective is to illustrate the ability of an Internet application to be integrated with real world application, for a seamless user experience. The system will allow a user to have a control over home devices and equipment, by manually sending a status update to Twitter, or by a click of buttons on our custom web-based interface. It is hoped that the final product can be used by group of people with tight life schedule, thus helping them in their daily chores.

### 1.4 Scope of Study

To develop the whole system in place, there are several skill sets required. The following table highlights some of the important scope of study:

Table 1: Scope of study for the project

Subject	Description
Web-based application development	Involve web interface design, backend process (including Twitter API) and other
Windows software development	Development of custom windows application to fetch data and interact with USB device.
USB interfacing and device development	Requires for PC-to-microcontroller interfacing in sending the data via USB
Basic wireless communication	Involve transmitting and receiving signal from one point to another point.
Microcontroller programming and circuitries	Requires for all parts of the project that involves microcontrollers.

For Final Year Project 1 (FYP1), several researches and studies has been conducted in order to identify the most suitable methods, technologies and techniques to be employed into the project. Evaluations on each of them are performed to choose the most suitable ones to be applied in the later stages, especially the development phase. Once the suitable methods, technologies and techniques are acquired, further exploration on the subjects is performed to start working on the real prototype system.

In this second stage, the Final Year Project 2 (FYP2), all the outcomes and information obtained from FYP1 are gathered and used for the development stage. The requirement of the project is set and all the components are developed and tested.

## **1.5 Report Organization**

In this report, there are five major chapters included. The first section is all about the introduction of the project and the Twitter model itself. The project objectives, problem statements and the scope of study were also included. In the second chapter, several literature reviews are conducted on the existing technologies that are related to the project.

In chapter 3, discussions will be conducted in detail on the methods, technologies and techniques that has been used throughout the project. It covers four major components of the project, which are the Software, USB Device, wireless transmission and the small-scale house model. Sample code snippets and figures are also included along with the explanations.

Chapter 4 highlights the results and discussions of the project - where the final product of the project is included. It also covers some basic understanding on how the whole system works and what are the requirements needed. The results from test cases conducted on the prototype are also included.

The final part of this report includes the overall conclusion of the report and the project itself. Several recommendations for future works are also included for further exploration and potential improvements.

## **CHAPTER 2**

### **LITERATURE REVIEW**

Home automation is a common subject anywhere in the world. A lot of companies come out with their own version of home automation system that make use of different type of medium or communication system, such as wireless remote or controller, internet connectivity and other. Currently, there is no home automation system that uses Twitter, or any web services for that matter, available in the market. In other word, this type of system is not developed on a mass scale to be shipped to the end users.

#### **2.1 Twitter Application Programming Interface**

Since the main highlight of this project is the use of Twitter in a non-conventional way, it is important to know the flexibility of this web application in integrating it into a real world application. Twitter Application Programming Interface (API) is one of the tools that is going to be used, which actually enables the realization of the initial idea for the project. With this API, it is possible to access the Twitter infrastructure in a third-party developed application to send data to and to fetch data from.

Twitter API documentation on its developer page provide all the necessary information for developers. It is safe to say that with this API, Twitter manage to expand the visibility of their service with various applications developed by third party developers. One of the API used in the project is user timeline resources.

From the documentation, it is given that the URL for this resource is

```
http://api.twitter.com/version/statuses/user_timeline.format
```

where the format can be in *json*, *xml*, *rss* or *atom*. There are also optional parameters that can be used along with above URL, such as *user\_id*, *count*, *page* and others. Therefore, for example if only the last 3 statuses is needed, the *count* parameter can be included as following:

```
http://api.twitter.com/1/statuses/user_timeline.json?count=3
```

They are of course more than this when it comes to API programming. Therefore, self-exploration is conducted in order to start working on an application for the project.

## **2.2 Microcontroller Programming**

Knowledge in microcontroller programming is inevitable as this project is fully microcontroller system based project. There are various types of programming languages that can be used to program a microcontroller; such as assembly and C. Although high flexibility can be achieved using assembly language, it is not as straight forward as C, especially for a new user. There are also different compiler applications available in the market - for example the freeware MPLAB IDE by Microchip, and commercial CCS C Compiler by CCS. Each compiler has their capabilities and built-in syntax and functions. However, this makes a program built in one compiler incompatible with the other compilers.

There are different types of microcontroller as well, depending on the capabilities, functionalities and the purposes of the application to be achieved. Microchip, for example, is one of microcontroller chip producers, which is quite popular among students and hobbyist. In this particular project, all the microcontrollers are Microchip based.

## **2.3 Real-life Twitter-based Applications**

There are some similar Twitter-based applications being developed by individuals on a DIY (Do It Yourself) project scale. These DIY projects are basically done as a proof-of-concept in illustrating the creative use of Twitter, as one of the most popular social networking service on the Internet, paired with an easy-to-use and flexible programming interface in their API, as explained earlier.

Basic idea of this project is inspired from free software called TweetMyMac (which was originally adapted from TweetMyPC). This software allows user to control their computers from the Twitter's direct messaging feature. Available commands include shutdown, grab the screenshot and get the current IP address [1]. Some of the commands can be very useful in the case where the computer is being stolen. However, it can only control the internal operation of the host computer - not any external applications or additional peripherals.

Botanicalls Twitter is a DIY project that also makes use of Twitter. Basically, a small kit of electronic circuit with some sensors is placed inside a vase of a real plant. This system will automatically send an update to its pre-defined Twitter account when it needs water, thus notifying the owner on his mobile phone [2].

There are several other small-scale projects conducted by individuals that can be relating to the interaction of Twitter with external hardware and peripherals. Most of them, however, involve one-way communication between the devices and the host computer.



## **2.4 USB-based Devices**

This project will also incorporate the use of Universal Serial Bus (USB) for the data transfer and transmission to/from the computer. Nowadays, USB-based peripherals are widely available in the market. The commonly use includes keyboard, mouse and mass storage device. There are a lot of USB-based device developments resources available, for example in [3].

In order to develop an USB-based device, ones need to learn and understand the basic knowledge of USB Protocol. This protocol outlines how the data transfer is done between the host, the hub, as well as the device itself. Error detection and correction, control flow and other important features are important in the protocol. Documentations on this protocol can be acquired from the official USB website for free to help independent developers around the world in developing their custom devices. There are also several open source USB development frameworks available for hobbyist and student to ease their device development. In this report, information regarding USB device development is included in Section 3.6 and 3.7.

## **2.5 Home Automation Technologies**

Various kind of home automation technologies are available in the market. The differences between these technologies are basically depends on how the home is actually controlled, and the level of flexibility among them. Some technologies use Internet connectivity, small-range remote control, mobile communication and many more. In this project, Internet connectivity, along with mobile communication is used as the control mechanism, but they are specifically integrated into the Twitter service. This Twitter integration is done with the help of Twitter API, which will be explained in later chapter of this report.

## CHAPTER 3

### METHODOLOGY

The research methodology for this project is mostly done by reading and self-exploration on various matters related to the technical skills, knowledge and tools (resources) required to perform the tasks in completing all the components needed for the project. Internet is generally the major resource for the self-exploration. The following sub-sections highlight the important components and their development process for the project.

#### 3.1 Project Development Components

This project can be divided into four (4) major components. By having this component-based development, it will ease the project management during the development stage as the explorations and development for these components can be performed in parallel. The major components are illustrated in the following figure.

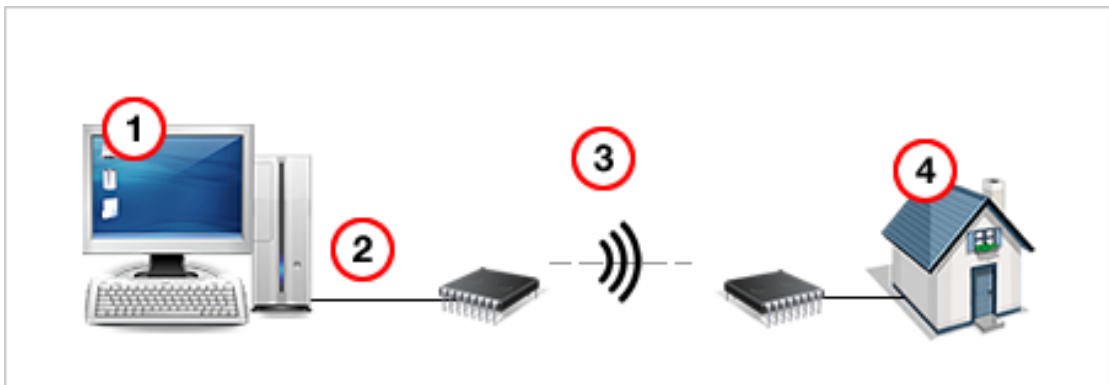


Figure 1: Four major components involved in the project

## **Component 1: Software**

The first component deals with the software part of the project, which mainly resides in a host computer. Several applications has been developed for the host computer in order to perform the following actions:

- (a) Communicate with Twitter via the Internet
- (b) Parse the Twitter status update from XML format to readable format by removing the unnecessary information
- (c) Periodically send the information to external microcontroller via USB connectivity.

The main requirement for the software part is to make sure it is able to continuously fetch the latest status updates from Twitter. This is important as to make sure the instructions sent by user will propagate through the components in the system as fast as it can. Detail discussion on this component is explained in **Section 3.3** for Web-based Interface, **Section 3.4** for Status Fetcher and XML Parser; and **Section 3.5** for Custom Windows Application.

## **Component 2: PC-to-Microcontroller interface**

A continuous data transfer from PC is needed to transfer the data fetched from Twitter (from Component 1) into the microcontroller circuit. Several options to achieve this process have been explored during the research stage, including serial port, parallel port and Universal Serial Bus (USB). For this particular project, USB has been chosen due to some of its advantages. Detail discussion on the options comparisons and USB device development is explained in **Section 3.6** for Hardware and **Section 3.7** for Firmware.

### **Component 3: Wireless communication**

A wireless transmission mechanism is required establish the communication between the first microcontroller which is connected to the host computer to the second microcontroller circuit that will be linked to various home devices and applications. Several options for this mechanism have been explored during the research stage. However, for this project, XBee Wireless Module has been chosen due to its low cost and low power consumption value. Detail discussion regarding the options and XBee Wireless Module is provided in **Section 3.8**.

### **Component 4: Home devices for automation**

The last component of the project is the various home devices and application that can be used to illustrate the home automation ability. It is not a primary part of the product itself, but it is added to the project development to be used for demonstration purposes. However, the project will only exhibit several simple automation possibilities, such as turning the lights ON and OFF, closing and opening a small-scaled door, and others pre-defined actions. More explanation regarding this component is explained in **Section 3.9**.

## **3.2 Project Development Stages**

As explained earlier, the whole project development is divided into four major components. This project division is also reflected on the flowchart. The approach is to have a parallel development on all the components. It is possible as each of the components is independent of each other, particularly during the development stage. This can reduce the lag time in the development phase as it can be carried out in parallel at the same time. Overall integration of the components is only performed at the end of the project for testing and troubleshooting purposes.

Overall project development stages are illustrated in the form of a flowchart in the next figure.

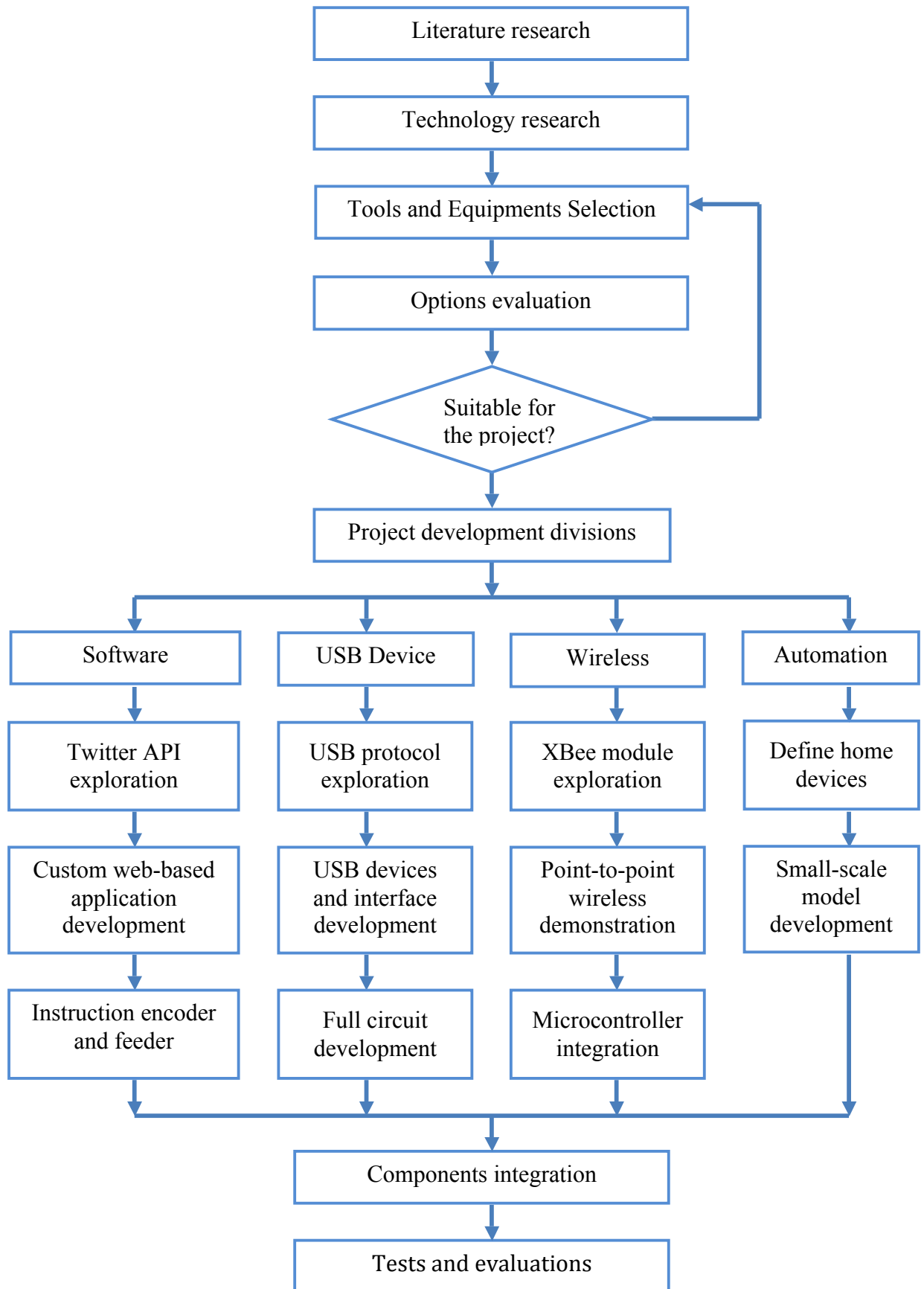


Figure 2: Overall project development stages

### 3.3 Web-based Application

#### 3.3.1 Introduction

One of the main parts in the software component is a custom web based interface. This web-based interface acts as a centralized control panel and one of the direct interaction mediums between the user and the system. Some of the features included onto the web-based interface are:

- Secure user authentication for proof of validity using OAuth
- Ability to send instruction via a click of a button
- Check recent instructions sent to the system

This web-based interface is developed from scratch using combination of several programming languages as highlighted in the following table, along with Twitter API and OAuth technology (will be explained in Section 3.3.2).

Table 2: Programming languages used in web-based application development

Languages	Purpose description
xHTML and CSS	For the front-end development of the interface that will determine the appearance of the website
PHP	<ul style="list-style-type: none"><li>• For the backend of the website</li><li>• To interface with the Twitter API and libraries in order to send and retrieve status updates as instructions</li><li>• To interface with the OAuth system for user validation.</li></ul>
JavaScript	To handle other eye candy appearance on the website

### 3.3.2 *Twitter API and OAuth authentication*

The Application Programming Interface (API) for Twitter is publicly available on their API website. Their approach of making the API available for the developers has enabled a lot of independent developers to develop various kinds of extended applications that incorporate Twitter's services - in web-based environment, standalone desktop applications for multiple operating systems (Microsoft Windows, Macintosh and Linux) and mobile application (iOS, Android, Symbian and others).

In this project, we are required to use this API in order to perform the following actions: (i) post status update, (ii) fetch status updates, and (iii) display current status updates. A simple example of getting a list of status updates from a user, using basic PHP is shown below:

```
require "twitter.lib.php";
$username = "username";
$password = "password";
$twitter = new Twitter($username, $password);
$xml      = $twitter->getPublicTimeline();
$twitter_status = new SimpleXMLElement($xml);
foreach($twitter_status->status as $status){
    echo $status->text; }
```

In this case, username and password of the user need to be supplied in order to use the code. This is not favorable as it can cause misused of user credentials by the developers. That is the major reason why we decided to use OAuth authentication method – where all the login credentials will be handled by Twitter themselves and the user will be able to allow or deny any access by custom applications without supplying any sensitive information.

OAuth is an open protocol to allow secure API authorization, which is officially adapted by Twitter [4]. By enabling this user authentication method, it will increase the security feature of the application, give more control to the user and simplify the process of sending instruction to the whole system.

To use OAuth, we have to register this custom application with Twitter in order to obtain the API key, consumer key and consumer secret key. This information will be used inside the coding. For this particular project, we registered the application as MicroAuto, as shown below:

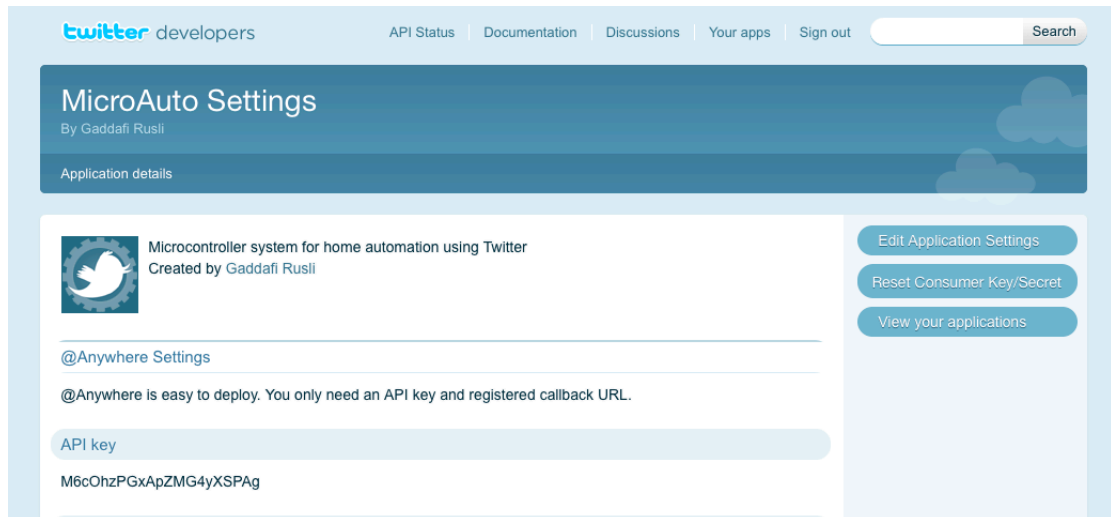


Figure 3: Setting page for the custom application registered with Twitter

When a user try to make a connection to MicroAuto, they will be prompted by Twitter, where they have to choose whether to allow MicroAuto to access their login credentials and perform other actions on their Twitter account, or deny the connection.

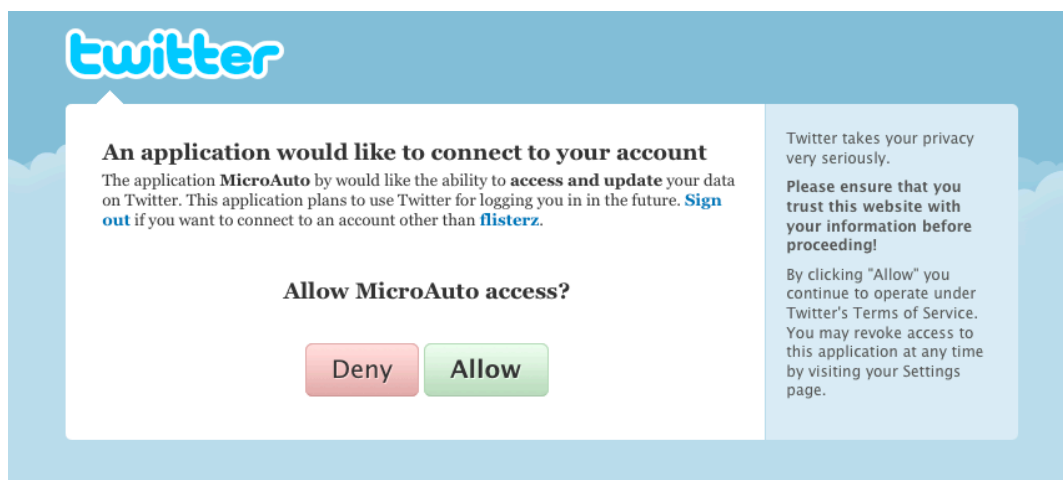


Figure 4: Twitter user will be prompted for external access



Some of the important code snippets are shown below with some brief explanation. Compared to previous code snippet, the following snippet doesn't require the username and password to be hard-coded into the coding [5]. This is because it is using the OAuth authentication method. The credentials are automatically fetched from Twitter when the user allows the access, and once it is verified, it will display the user's name on the screen.

```
/* Connect to the Twitter API */
$to = new TwitterOAuth($consumer_key, $consumer_secret,
$_SESSION['oauth_access_token'],
$_SESSION['oauth_access_token_secret']);
$content = $to-> OAuthRequest
('https://twitter.com/account/verify_credentials.xml',
array(), 'GET');
$user = simplexml_load_string($content);
if ($user->screen_name!='') {
    echo 'Hello, '.$user->screen_name.'
```

The following code snippet, then, can be used to send an update to twitter to be used as instruction to the system. It can either be in a form of textual hyperlinks or visual buttons. This will be used as the mechanism to send instructions to the system.

```
$content = simplexml_load_string($to->
OAuthRequest('https://twitter.com/statuses/update.xml',
array('status' => This is the instruction to be sent'),
'POST'));
```

The complete source code files used for this web-based interface, including TwitterOAuth library, are included in **Appendix A**.

### ***3.3.3 Overcoming duplication limitation***

During the development stage, it is discovered that Twitter imposes status duplication limitation on the status updates send by the users. It works by checking the last 10 status updates to see if the user already posted the same exact phrase. This introduced a little problem for the project, as users need to send a pre-defined phrase, which will be the same every time, for each action. In order to overcome this limitation, we decided to add extra characters at every phrase that are to be send to Twitter. These extra characters consist of a forward slash (/) and two random characters. For example:

LIGHTS ON / 34

These random characters can be anything, either in numerical or alphabetical form. This method will help to differentiate each status updates, so Twitter will not treat them as duplicates. As for the web-based interface, it will automatically add the random characters when users click on the buttons. However, for manual status update, user has to manually insert them at the end of each phrase.

### ***3.3.4 User interface design***

The web-based application development is completed with a usable interface design that makes it easier for the user to access the application itself. This interface will be responsible in guiding the user from the login to the sending of instruction via the click of buttons.

The visual and graphic in the interface is designed using Adobe Fireworks CS5, and was converted into a full working web interface using xHTML and CSS languages. These languages are mixed together with the backend code, along with the Twitter API in order to make it fully functional. The source code files are included in **Appendix A**. Screenshots of the web interface are included in **Appendix B**.

## 3.4 Status Fetcher and XML Parser

### 3.4.1 Introduction

Other than the web-based application, which is hosted on the Internet, another web-based page is needed. The objective of this small application is to read the latest status from a pre-defined Twitter account, and parse all the unnecessary information (such as the status unique ID and the date) to only capture the needed data (status text). This one-page status fetcher can be placed in the host computer with local server (*localhost*) enabled.

### 3.4.2 XML parser

One of the file formats that are provided by Twitter in their API is XML (extensible Markup Language), other than JSON. For this fetcher, we are fetching the status using the XML formatted information. The following screenshot shows a portion of the XML-formatted information of a single status. Full screenshot can be seen in **Appendix C**.

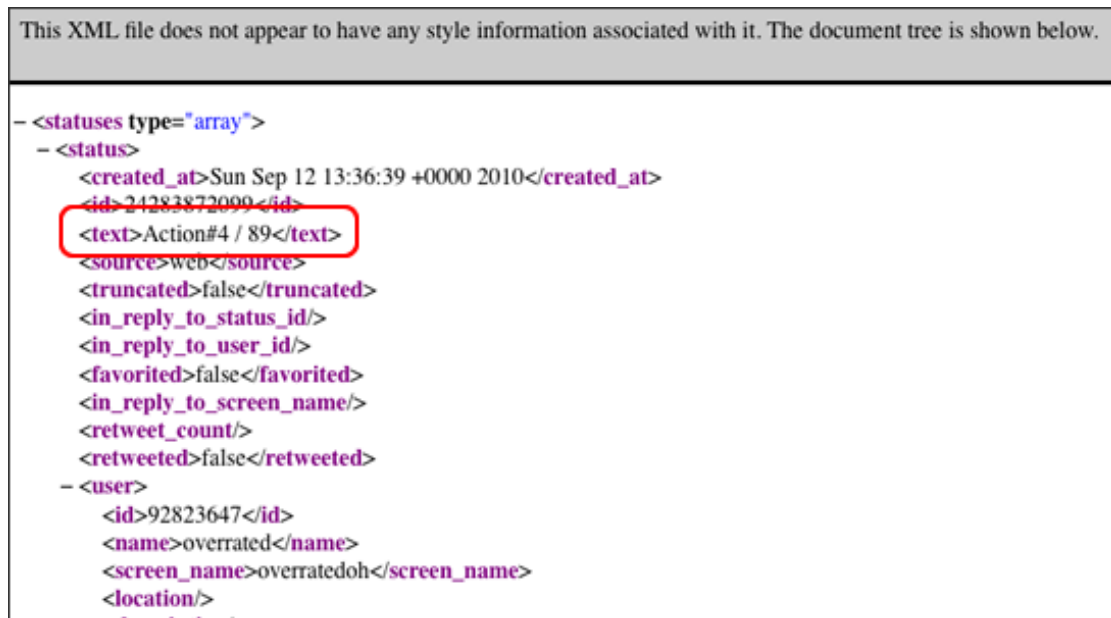


Figure 5: XML-formatted information from a single status update

The XML parser will be applied onto this XML-formatted page to grab the status text (as highlighted with the red box in Figure 5).

```
$xml = simplexml_load_file($source);
$raw = $xml->status->text;
if (strpos($raw, '/') !== false) {
    $content = "<span>".$raw."</span>";
    $actBeg = strpos($content, '<span>', 0);
    $actMid = substr($content, $actBeg+6);
    $actEnd = strpos($actMid, ' /');
    $action = substr($post, 0, $actEnd);
}else{
    $action = $raw;
}
echo $action;
file_put_contents("action.txt",$action);
```

The *simplexml\_load\_file()* function is used to read the XML-formatted file generated by Twitter. Looking at the portion of XML file in Figure 5, it is shown that the *text* is a child of *status*. Therefore, in order to obtain just the content of *text*, the following snippet is applied:

```
$raw->status->text
```

The rest of the code is basically to get rid of the random characters generated by the web-based application, which is needed to overcome the duplication limitation imposed by Twitter, as explained in **Section 3.3.3**.

Finally, the status text that has been extracted is stored inside a text file called *action.txt* using the *file\_put\_content()* function. The file name is arbitrary, but need to be defined in the custom Windows application, which will be explained later. Full source code for this part is included in **Appendix D**.

## 3.5 Custom Windows Application

### 3.5.1 Introduction

The major part of software component is the development of custom Windows application, which is located in the host computer. This custom Windows application works in pair with the custom USB device hardware and firmware. Generally, this application acts as the intermediate between the data fetched from Twitter via the Internet, with the real-life hardware connected via USB.

The development of this custom application is conducted in Microsoft Visual Studio 2008 using C# language and Windows API. This development environment allows us to create an application with simple Graphical User Interface (GUI) in a much easier way. Furthermore, this custom application doesn't have any complicated features other than reading and sending some simple information via USB. Full source code that is used for this custom Windows application development is included in **Appendix E**.

### 3.5.2 USB connectivity

Some important files and source code needed for USB connectivity in this custom Windows application are obtained from sample codes provided by Microchip in their Microchip Application Library (to be introduced further in **Section 3.7.2**). From **Appendix E**, the second file (*usb\_interface.cs*) is the necessary file involved in USB connectivity with the USB device. It involves USB API interface and assigning the Vendor ID (VID) as well as the Product ID (PID). Since registration and payment are required to obtain unique VID and PID, the default IDs provided by Microchip in their sample file are used for this project.

This file is also involved in the opening and closing of pipe. In the USB protocol, the concept of pipe represents a logical connection between the software on the host computer and the endpoint on the USB device. It acts as a communication channel that have a set of parameters associated with them such as how much bandwidth is allocated to it, what transfer type (Control, Bulk, Isochronous or Interrupt) it uses, direction of data flow and maximum packet/buffer sizes.

### 3.5.3 Actions and interval definitions

Before the application can be used to send instruction to the USB device, a list of actions is pre-defined inside the application. These pre-defined actions will be used by the application to determine which instruction is to be transferred to the USB device, based on the status updates (or instruction) sent by the user via Twitter. The list is defined on the top portion of *MicroAuto.cs* (refer to **Appendix E**). The phrase on the right side is defined to be the exact phrase (case sensitive) that is going to be used by the user to update their Twitter status.

```
//define the read interval - in seconds
public int second = 1;

// instruction/action definition
public string action1 = "LCD ON";
public string action2 = "LCD OFF";
public string action3 = "DOOR OPEN";
public string action4 = "DOOR CLOSE";
public string action5 = "LIGHTS ON";
public string action6 = "LIGHTS OFF";
public string action7 = "FAN ON";
public string action8 = "FAN OFF";
```

. For example, if the user wants to turn ON the lights, the exact phrase of LIGHTS ON (along with the extra characters as explained in **Section 3.3.3**) must be sent to Twitter. The application will decode the phrase, and identify that it is belong to the string called *action5*, and the further process will be conducted from there.

Above code snippet also shows the interval definition, set to 1 second. This interval is the interval used by the custom application to read/fetch content from the text file (to be further explained in **Section 3.5.4**). In this case, the application will read the content of the text file every 1 second.

### 3.5.4 Periodic content fetch from text file

One of the features in this custom application is to periodically read the content of the text file generated by the status fetcher as explained in **Section 3.4**. This feature is important in order to get the latest instruction send by the user to be sent to the USB device, via the USB connectivity function built inside this application. It also has to be done periodically every  $n$  seconds, where  $n$  can be any value, to simulate the real-time functionality of the system.

The following code snippet, taken from the `MicroAuto.cs` file, shows the part where the application read the text file.

```
public void Read_file() {
    StreamReader textFile = new StreamReader("action.txt");
    string fileread = textFile.ReadToEnd();
    if (fileread == action1)
    {
        ReadAction.Text = action1;
        usb_int.actionSend(1, true);
    }
    else if (fileread == action2)
    {
        ReadAction.Text = action2;
        usb_int.actionSend(2, true);
    }
    ....
}
```

The snippet above shows that the application is using the `StreamReader()` function to read the content of text file. The content is stored inside a string called *fileread*. This string is then compared with the pre-defined actions. If the content of the text file is comparable, or available inside the pre-defined list, a specific data is send to the USB device accordingly. Otherwise, an error message is displayed on the application.

The value “1” or “2” inside the *usb\_int.actionSend()* function is the one that being send to the USB device through the pipes for further processing. This value will enable the system to differentiate between actions and devices to be controlled.

In order to make the read functionality to work automatically at all time, it has to be executed periodically in a small interval of time. A simple timer function is created inside the application to handle the interval sequence. The code snippet below is taken from *MicroAuto.cs*, showing several portion of code required for the timer function.

```
private System.Windows.Forms.Timer timer1;
...
this.timer1.Enabled = true;
this.timer1.Interval = 1000;
this.timer1.Tick += new
System.EventHandler(this.timer1_Tick);
...
private void timer1_Tick(object sender, EventArgs e) {
    if (interval < 1) {
        interval = second;
        Read_file();
    } else {
        interval -= 1;
    }
    lblSec.Text = interval.ToString() + " seconds";
}
```

The timer works in a second by second basis, as noted by the 1000 integer, which represents 1000 milliseconds, which is equal to 1 second. The amount of interval is defined earlier along with the action definition (refer to **Section 3.5.3**). The *timer1\_Tick()* functions works by subtracting 1 from the interval value. Once



the interval is equal to zero, it will reset the timer and re-execute the *Read\_file()* function to obtain the latest content from that file. By doing this, the system will always get the recent actions/instruction sent by users. The value of the timer countdown is also displayed on the application.

### 3.5.5 User interface development

The custom application will be running on the host computer at all time. Therefore, it is important to come out with a very simple application with simple Graphical User Interface (GUI) as not to confuse the user with any complexities. The application should also have a small footprint on the host computer's resources and memory allocation.

The custom application for this project, called MicroAuto (to complement the web-based application), is developed using Microsoft Visual Studio in C# language. It is safe to say this development environment allows for a faster and easier approach in dealing with GUI. The following figure shows the development environment.

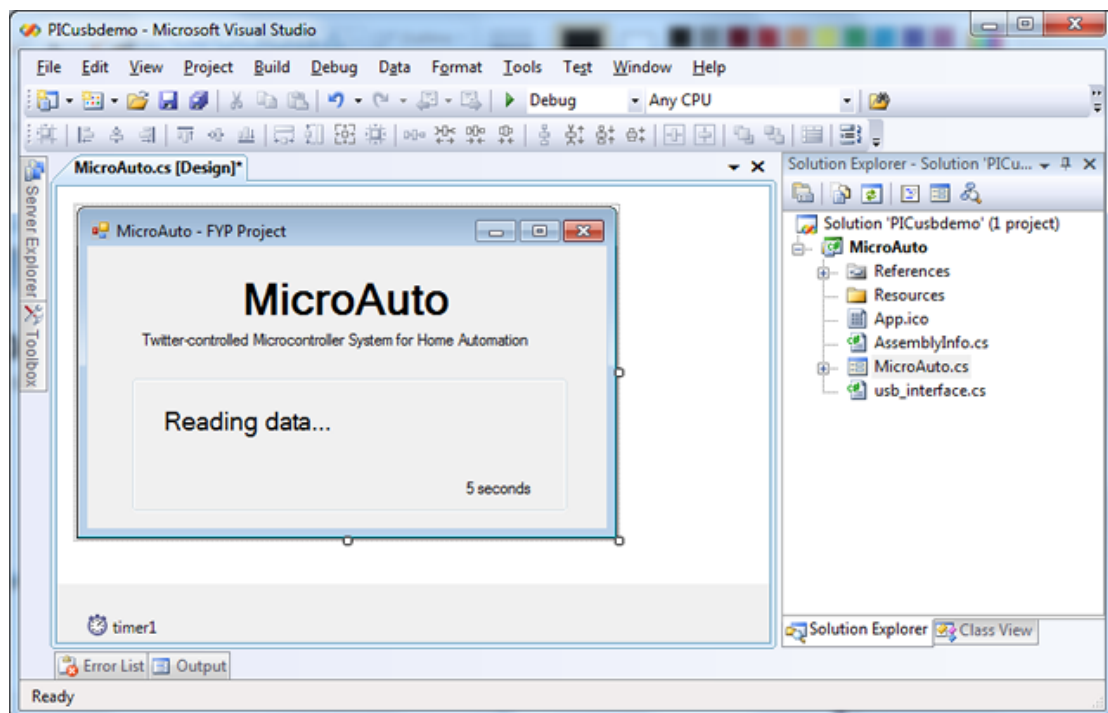


Figure 6: Visual Studio development environment

## 3.6 USB Device Development: Hardware

### 3.6.1 Introduction

Nowadays, USB is used in almost all external peripherals that need to be connected to a computer. Other than being plug-and-play, it is also a fast, bi-directional, low-cost, dynamically attachable serial interface that is consistent with the requirements of the computer platform now and future. It is also a popular option for manufacturer being its primary advantages of having the ability to power larger circuits and higher connection bandwidth of the connection as explained in [3].

The following table summarize some of the features comparison between USB, Serial and Parallel port [5]. As shown in the table, it is believed that – moving forward - USB is the most suitable technology to be used in interfacing the device with the host computer.

Table 3: Comparison between USB, Serial and Parallel port

	USB	Serial	Parallel
Industry standard	Yes	Yes	No
Bandwidth	12 Mbps USB 2.0 – 480Mbps	115 Kbps	115 Kbps EPP/ECP – 3 Mbps
Number of devices	127 on a single USB bus	Limited to number of ports available on the computer	Limited to number of ports available on the computer
Bus Power	Up to 500 mA at 5V	No	No
Cable Length Limit	5m / 16.4ft	3m / 10ft	1.8m / 6ft
Plug & Play	Yes	No	No
Hot Swappable	Yes	No	No

A typical USB system consist of the following parts

- One host - responsible to deal with the protocol and control the media access to the USB bus, for example the host computer.

- b. Hub - responsible for detecting an attachment and detachment of devices and handling the power management for devices from the bus.
- c. Device - in this project, the microcontroller circuits can be considered as the device. It can be self-powered or bus-powered.

### 3.6.2 Transfer types

As specified by the USB specifications, there are four transfer types. Brief description for each transfer types is included in the following table. It also specifies which USB device type (full-speed or low-speed) is supported by each transfer type.

Table 4: USB transfer types

Transfer type	Description	USB device type support	
		Full-speed	Low-speed
Isochronous	Provide transfer method for large amount of data up to 1023 bytes. Timely delivered ensured but data integrity not ensured.	●	
Bulk	Provide transfer method for large amount of data with ensured data integrity. Delivery timeliness is not ensured.	●	
Interrupt	Provide transfer for small blocks of data with ensured delivery timeliness and data integrity	●	●
Control	Provide transfer for device setup control	●	●

For this particular project, only small blocks of data are to be transmitted via the USB connection, therefore the Interrupt transfer type is the most suitable method. In addition, it also has more advantages as the delivery timeliness and data integrity are ensured.

### 3.6.3 Microcontroller circuit

During the exploration and research stage of the project, a comparison between several microcontroller is conducted in order to choose the most suitable microcontroller for the project, especially between the 16F877A and 18F4550 microcontrollers. Based on the result of that comparison, a decision is made to use the 18F4550 PIC microcontroller for this particular project. One of the major reasons is because of its built-in USB capability that is required to develop a custom USB device in a much easier and faster approach.

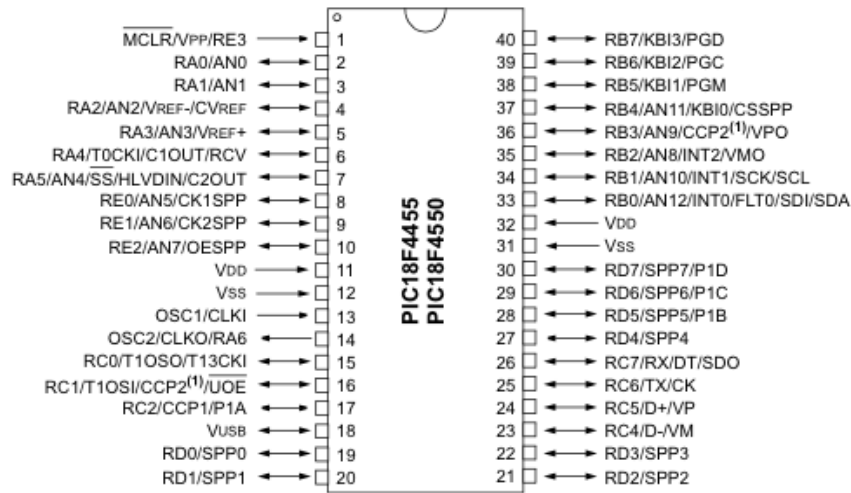


Figure 7: Pin-out of PIC18F4550

Based on the datasheet provided by Microchip, The PIC18FX455/X550 device family contains a full-speed and low-speed compatible USB Serial Interface Engine (SIE) that allows fast communication between any USB host and the PIC microcontroller. To make use of its SIE feature, it can be directly interfaced with the USB device, utilizing the internal transceiver. Optionally, external transceiver can also be used, but it is not going to be adapted for this project. Other than internal transceiver, this microprocessor also has a built-in 3.3V regulator as a power supply for the transceiver [7].

Figure below shows some of the important components related to USB device developments that are available in PIC 18F4550. As the figure shows, the USB SIE, transceiver and the voltage regular are located inside the microprocessor (on the left side of the dashed line).

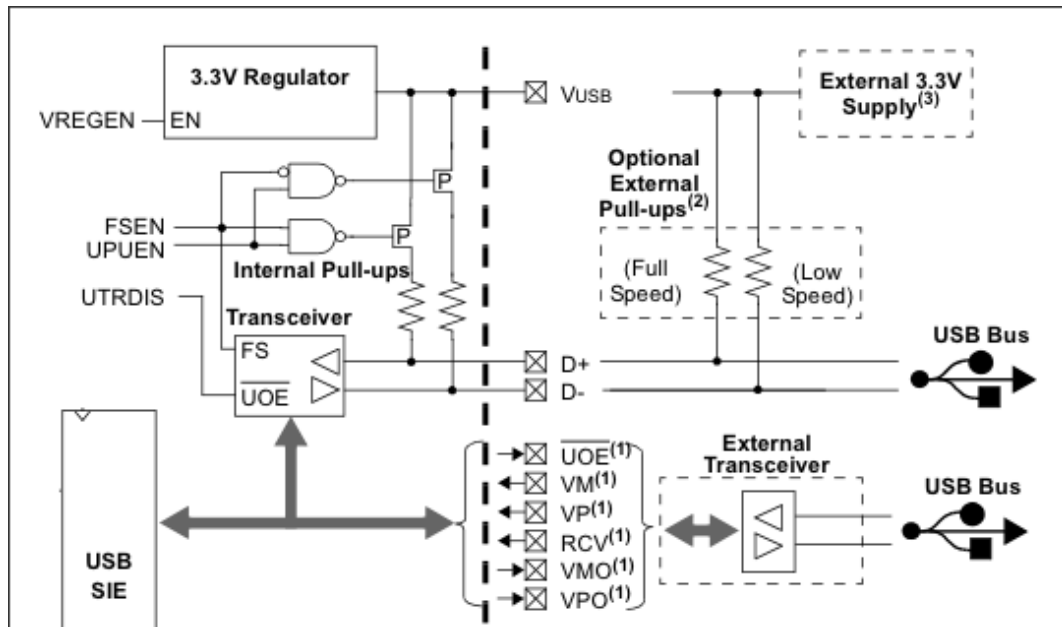


Figure 8: Built-in USB components in PIC 18F4550

Figure 9 shows the basic schematic for the USB device used in this project. Note that the USB socket is connected to Pin 23 (D-) and Pin 24 (D+), which are used for the data transfer with the host computer once it is connected. Also note that the LEDs connected to PORT D of the microcontroller is arbitrary, as they are only used for testing purposes during the development stage.

It is also shown from the schematic that the USB socket is also the source of power supply for the device. Therefore, the device is a passive device, as it does not need to have an external dedicated power supply in order to function. This is important as it is much favorable by the user and it will reduce the complexity of the whole system.

Full schematic for the USB device along with the wireless transmission component (to be explained in **Section 3.8**) can be seen in **Appendix F**.

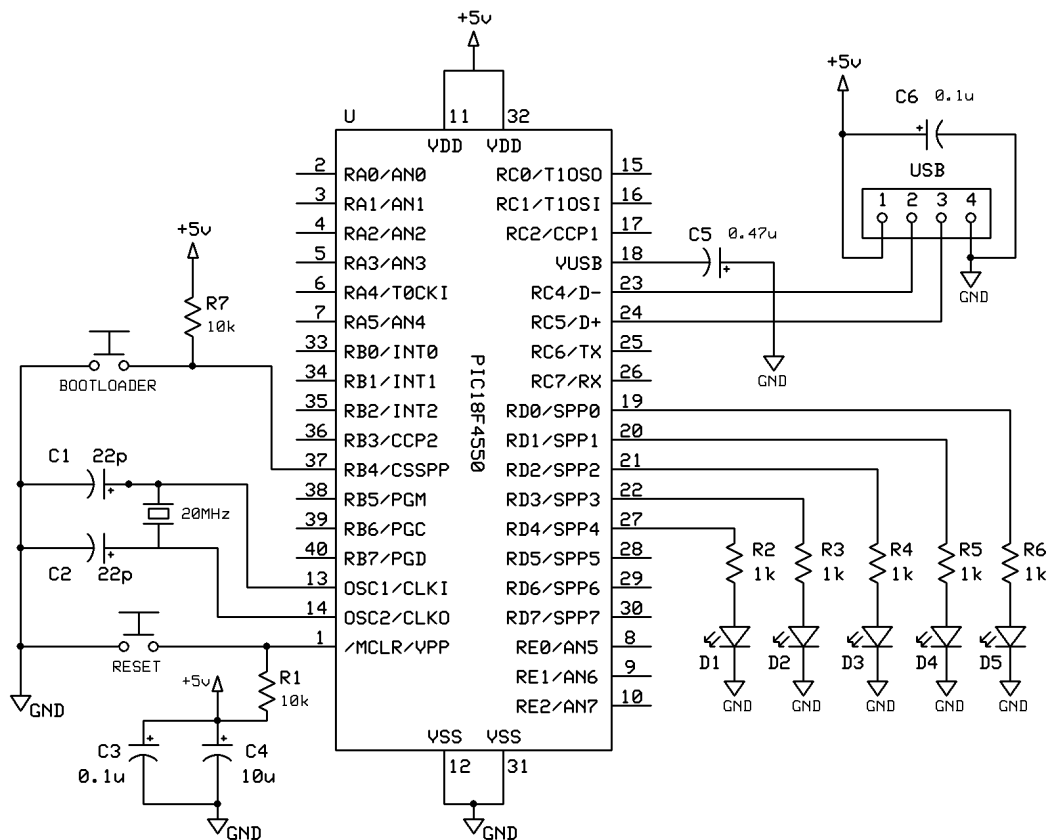


Figure 9: Simple circuit schematic for USB device with 18F4550

There are four major type of USB ports and connectors – namely Type A, Type B, Mini-A and Mini-B – as shown in the following figure (refer to **Appendix G** for more comparisons data).



Figure 10: Four types of USB connector.

From left: Type A, Type B, Mini-A and Mini-B

For this particular project, Type A connector is used, as it is the most common type available in most computers and portable devices. The pins and wires (color) for Type A USB connector is shown in the following table.

Table 5: Pins for Type A USB connector

Pin	Signal	Color	Description
1	Vcc	Red	+5V
2	D-	White	Data -
3	D+	Green	Data +
4	Ground	Black	Ground

The color property stated in above table is referring to the color of the wire inside a typical USB cable. The combination of Red and Black wires is enough to supply a sufficient power supply of 5V for the device.

#### ***3.6.4 Converting to Printed Circuit Board***

During the development stage of the USB device, a typical breadboard is used for the circuitries to enable flexible troubleshooting and expansion. Once the device is proven to works after several tests, the prototype is converted to a printed circuit board (PCB). The PCB design is done using a specialize software, Eagle Layout Editor, by CadSoft. The fabrication process is performed in the laboratory, assisted by the lab technician. However, due to facility limitation, only single layer PCB can be produced. Plus, the conversion to PCB only reduced the size to half of the original size of breadboard prototype. Photo of the completed PCB circuit can be seen in **Section 4.1**.

## **3.7 USB Device Developments: Firmware**

### ***3.7.1 Introduction***

The firmware of USB device will be the operating brain of the custom USB device hardware. The firmware is programmed into the PIC 18F4550 microcontroller. This firmware is responsible for several tasks including establishing communication with the host computer, getting data from the host computer, sending data to transmitter circuit, and others.

### ***3.7.2 Microchip Application Library – USB Framework***

The development process of this firmware is assisted by the availability of a free USB Framework and sample codes provided by Microchip. This framework is a part of Microchips Application Library (MAL) - a collection of Microchip firmware libraries and demo projects - which is available for free on their website. Other than for USB development, this comprehensive application library also includes the following libraries: Graphics Library, Memory Disk Drive, TCP/IP Stack, mTouch Capacitive Touch Library, and Smart Card Library. By the time of this report is written, the latest version of MAL is released on 4th August 2010 (Microchip Application Libraries v2010-08-04), which is the exact version that we are using for this project.

This USB framework support USB on 8-bit, 16-bit and 32-bit microcontrollers. The source files provided in the framework are royalty free, which means we can use them without having to pay anything to Microchip. It includes USB firmware for the microcontroller as well as a USB device driver for the PC, which allows the PC to treat the microcontroller as a USB device. Classes supported include HID, CDC, MSD and generic.



### 3.7.3 Customizing the firmware

From the framework, several sample codes have been used as the based for our USB project. This project comes with several source codes that are vital in establishing the connection between the USB device and the host computer. As far our purpose is concern, only some of the files require modification in order to achieve our desired needs on the firmware. Other than adding additional piece of code for the customization, some original piece of code that are not necessary were also removed.

### 3.7.4 I/O configuration

In the USB device, the main microcontroller that handling the USB connectivity will receive the data via the data lines, which are D- and D+ pins (refer to **Section 3.6.3**). It doesn't have any other inputs other than via this data line. In the I/O configuration, the output has to be defined to set the ports and pins required for outputting the data. The following snippet shows some of the configurations.

```
#define mInitAllLEDs()      LATD &= 0x00; TRISD &= 0x00;
                           LATB &= 0x00;  TRISB &= 0x00;

#define mLED_A              LATBbits.LATB0
#define mLED_B              LATBbits.LATB1

#define mLED_1              LATDbits.LATD0
#define mLED_2              LATDbits.LATD1
#define mLED_3              LATDbits.LATD2
#define mLED_4              LATDbits.LATD3
#define mLED_5              LATDbits.LATD4
#define mLED_6              LATDbits.LATD5
#define mLED_7              LATDbits.LATD6
#define mLED_8              LATDbits.LATD7
....
```

The two *mLED\_A* and *mLED\_B* outputs are the two indicator lights used in this device to give the indication on the connectivity status based on their blinking pattern. The other 8 output pins is configure to send the 8-bit binary number to the XBee transmitter circuit based on the instruction received from the host computer.

As explained in **Section 3.5.4**, a value is sent by the custom Windows application inside the *usb\_int.actionSend()* function, in the form of single digit value, to the USB device. This single digit value received by the USB device will be translated into the unique 8-bit binary number in the following snippet.

```
if(dataPacket.led_num == 1) {
    mLED_1 = 1; mLED_2 = 0; mLED_3 = 0; mLED_4 = 0;
    mLED_5 = 0; mLED_6 = 0; mLED_7 = 0; mLED_8 = 0;
    counter = 0x01;
} else if(dataPacket.led_num == 2) {
    mLED_1 = 0; mLED_2 = 1; mLED_3 = 0; mLED_4 = 0;
    mLED_5 = 0; mLED_6 = 0; mLED_7 = 0; mLED_8 = 0;
    counter = 0x01;
...

```

The 8-bit binary number is unique to every instruction sent, and action to be performed. As the continuity from the action definition in **Section 3.5.3**, the binary digit is assigned to each action in the following table.

Table 6: Actions definitions and representations

Variable	Unique Phrase	Unique Value	Unique Binary
Action mapping in the custom Windows Application	Phrases to be used for Twitter status update	Single digit number send to USB device	Output from the microcontroller to XBee transmitter
action1	LCD ON	1	00000001
action2	LCD OFF	2	00000010
action3	DOOR OPEN	3	00000011
action4	DOOR CLOSE	4	00000100
action5	LIGHTS ON	5	00000101
action6	LIGHTS OFF	6	00000110
action7	FAN ON	7	00000111
action8	FAN OFF	8	00001000

## 3.8 Wireless Transmission

### 3.8.1 Introduction to XBee wireless module

In this project, wireless communication is used for data transmission between the first microcontroller, which is connected to the host computer, to the second microcontroller circuit, which is linked to various home devices and applications. In this case, wireless is preferred compared to wired system, as it is less messy and much easier to be set up on the user end.

Based on the research conducted in the earlier stage, XBee Wireless Module has been chosen as the means for wireless communication. This module uses IEEE 802.15.4 networking protocol for fast point-to-multipoint or peer-to-peer networking. It can form self-healing mesh networks - great for making a wireless control network that spans from one corner of the house to the other, which is adequate for the prototype development of this project. The following table provides some basic specification of an XBee Wireless Module [8].

Table 7: Basic specifications of XBee Wireless Module

Power output	1mW
Indoor/Urban range	Up to 30 m (100 ft)
Outdoor/RF line-of-sight range:	Up to 90 m (300 ft)
RF data rate	250 Kbps
Interface data rate	Up to 115.2 Kbps
Operating frequency	2.4 GHz
Receiver sensitivity	-92 dBm

### 3.8.2 Configurations setting

Before using an XBee module in the circuit, the configuration setting can be checked in order to see whether it has any issues or not. For that purpose, a simple application such as X-CTU can be used. In order to use this application, XBee module has to be connected to the host computer – for example via USB cable, using XBee Explorer USB board (refer to **Appendix I**). The following figure shows the configuration setting in X-CTU application.

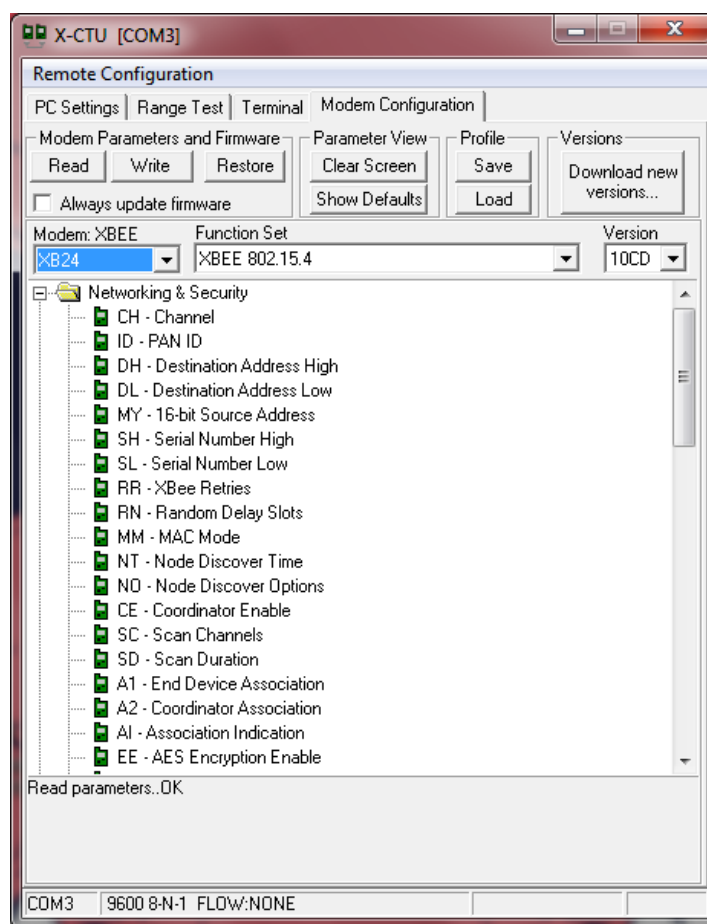


Figure 11: Configuring XBee using X-CTU

Note that this procedure is a one-time procedure only. Once the XBee functions as required, there is no need to use this method throughout the project phase, except for troubleshooting or firmware upgrade.

### 3.8.3 Wireless modules circuits

In this project, two XBee modules are required to perform a point-to-point transmission. The first one is connected to microcontroller 1 on the host computer side, and will act as the transmitter unit. The second module is connected to microcontroller 2 on the home devices side, and will act as the receiver unit. This scheme is illustrated in the following figure.



Figure 12: Illustration of wireless transmission configuration

The important and basic pins assignment of XBee module is shown in the following table. These four pins are the minimum requirement for a basic XBee connectivity to a PIC microcontroller. Pin 2 (DOUT) and Pin 3 (DIN) are connected to the Rx and Tx pin (on the microprocessor) respectively. In fact, these are the only pins we are using for this particular project.

Table 8: Four basic pin assignments on XBee Wireless Module

Pin	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / CONFIG	Input	UART Data In
10	GND	-	Ground

Note that the power supply for the module is 3.3V rather than 5.0V. Therefore, XBee Explorer Regulated (refer to **Appendix I**) is used for each XBee module, with built-in voltage regulator to provide the 3.3V power. The following figure shows the basic connection between the XBee module and the microcontroller, using only four pins detailed before. Full basic circuit schematics for transmitter and receiver are provided in **Appendix J**.

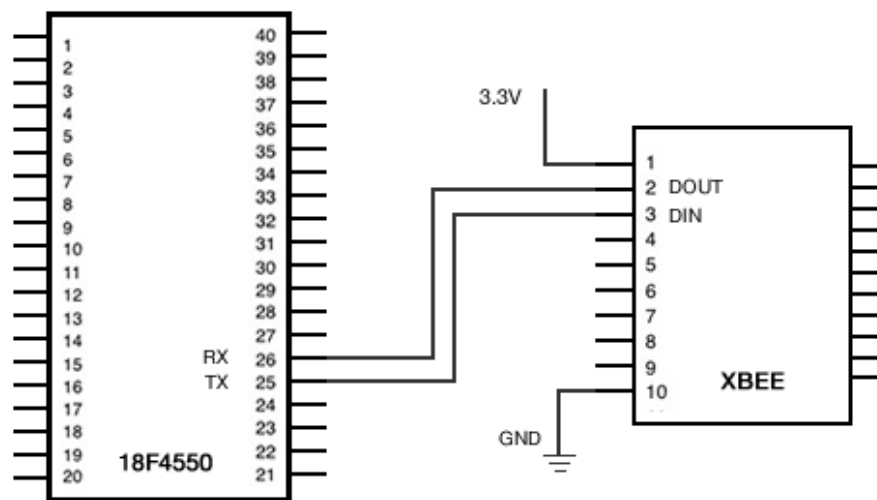


Figure 13: Basic four pins connection from XBee module to microcontroller

### 3.8.4 Transmitter and receiver

As explained earlier, the two XBee modules are used as a transmitter and a receiver. Compared to the other part of this project, the firmware of both transmitter and receiver is developed in CCS compiler. Although this compiler is essentially the same as MPLAB (using C language), CCS compiler has a much simpler syntax and approach for RS232 implementation, which is needed for XBee module.

For a quick illustration on the wireless communication component in this project, snippets from transmitter and receiver are explained below. In the following code snippet, which is taken from the transmitter side, if the switch at Pin A0 is initiated, it will send a signal “a”. Otherwise, it will send signal “b”.

```

while (TRUE) {
    if (input(PIN_A0)){
        printf("a"); //sends signal a
    } else{
        printf("b"); //sends signal b
    }
}

```

The `printf("a")` statement here means that the character “a” is *printed* through the RS232 protocol and it allows XBee module to transmit that particular character. On the receiver side, if the received signal is the character “a”, it will turn on the LED 0. Otherwise, it will turn it off.

```

while (TRUE) {
    if (x=='a'){
        output_high(LED_0);
    } else {
        output_low(LED_0);
    }
}

```

Full complete source code files used for the transmitter and receiver module are provided in **Appendix K**. These two source code files are compiled using CCS compiler.

The following figure shows a simple point-to-point transmission using the code explained earlier. The circuit labeled “A” is the transmitter while the circuit labeled “B” is the receiver. Flipping the switch in A trigger the signal transmission, and it will be fetched by B.



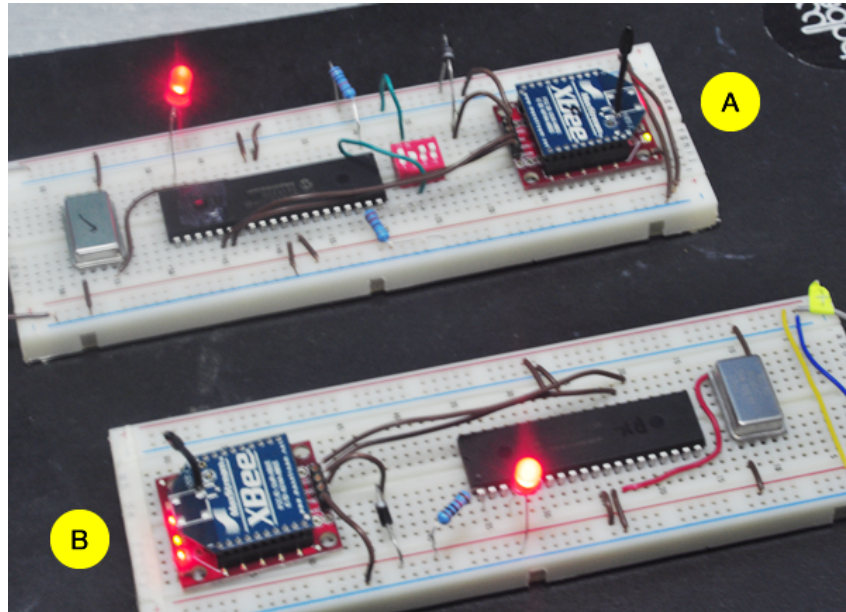


Figure 14: Point-to-point transmission demonstration from A (TX) to B (RX)

Note that in this particular project, XBee will only be used as a mean of transferring data from the first microcontroller to the second microcontroller. No processing or complex algorithmic operations will be carried out throughout the transmission.

Table 9: Designated character for each transmitter input

Transmitter Input (PORT B)	Transmitted Character	Receiver Output (PORT B)
00000001	a	Pin B0 = 1
00000010	b	Pin B0 = 0
00000011	c	Pin B1 = 1
00000100	d	Pin B1 = 0
00000101	e	Pin B2 = 1
00000110	f	Pin B2 = 0
00000111	g	Pin B3 = 1
00001000	h	Pin B3 = 0
00001001	i	Pin B4 = 1
00001010	j	Pin B4 = 0

### 3.9 Small-scale Home Devices Model

#### 3.9.1 Introduction

The final product out of the project is a fully working prototype that can be used to illustrate the working integration between all the major components. For the purpose of demonstration, a small-scale house with several ‘home devices’ is developed to represent the device automation. These ‘devices’ will be controlled by the system. Therefore, some actions need to be pre-defined on how to represent them in a small-scale model as to produce a good demonstration for the viewers. Several options and approaches that are considered is shown in the following table:

Table 10: Small-scale home device automation representations

Item	Operation	Implementation
Lights	Turning them ON and OFF	An array of super bright LEDs is used as the light source
Door	Opening and closing in sliding motion	A small servo motor is used to provide a simple mechanical movement
Fan	Turning it ON and OFF	A small DC motor with blade is used to illustrate the fan
Television	Display message on screen	A 16 x 2 characters LCD display is used as the display panel to display predefined characters/symbols

This initial outline might be expanded further - by adding more operations and mechanisms on the small-scale model. By having more example of device automation on the house model, it will reflect the efficiency and the real potential of the system.

### **3.9.2 *Microcontroller circuits***

In order to control the home devices, microcontroller has to be programmed to receive the input from the XBee receiver circuit and convert it to real action on the devices defined earlier – the lights, fan, door and television.

Full source code files are included in **Appendix M**. Note that the source codes included in the report are as of this writing – more application is to be added to the house model.

### **3.9.3 *House model***

Initially, the model of the house is to be built using Perspex material. However, after several re-considerations, it is decided that the model of the house is to be built using white foam board. Other than of its look, this material is chosen because it is lightweight, easy to handle, robust enough for its purpose, and easy to get.

The dimension of the house model is approximately 38cm x 44cm x 30cm. The two walls and floor are built in two layers in order to hide all the wirings and circuit under the house. This is to avoid distraction on the viewers so only the important part is viewable. Photo of the house model is shown in Figure 18 in **Section 4.1**.

## **CHAPTER 4**

### **RESULTS AND DISCUSSION**

After almost one year of duration given for Final Year Project, a working prototype has been successfully constructed. This working prototype are able to demonstrate the objective of the project – namely to illustrate the ability of Internet application to be connected to the real world environment. Based on the several testing phases conducted on the prototype, it is safe to say that it is functioning as intended.

#### **4.1 Full System Prototype**

The whole system of the project consists of software and hardware components. Therefore, it is not possible to conclude the working prototype of the project as a single physical item. Furthermore, it is not safe to say that the fourth component of the project, which is the small-scale home devices model, is not exactly a part of the real outcome of the project. It is merely a medium of illustration and demonstration purposes, except that it is using the receiver circuit, a portion of wireless transmission component. Overall, a complete set of working prototype for this project consists of:

- i) Web-based application interface (refer **Appendix B**)
- ii) Custom Windows application
- iii) USB device
- iv) Wireless receiver module
- v) Small-scale house devices model

The following figures shows some screenshots and photos that are related to the working prototype of this project.

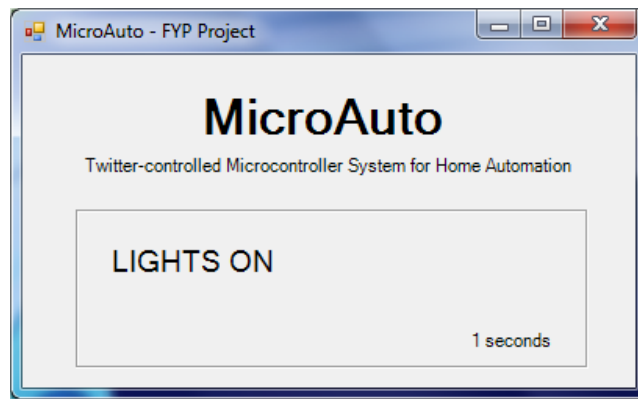


Figure 15: Screenshot of the custom Windows application

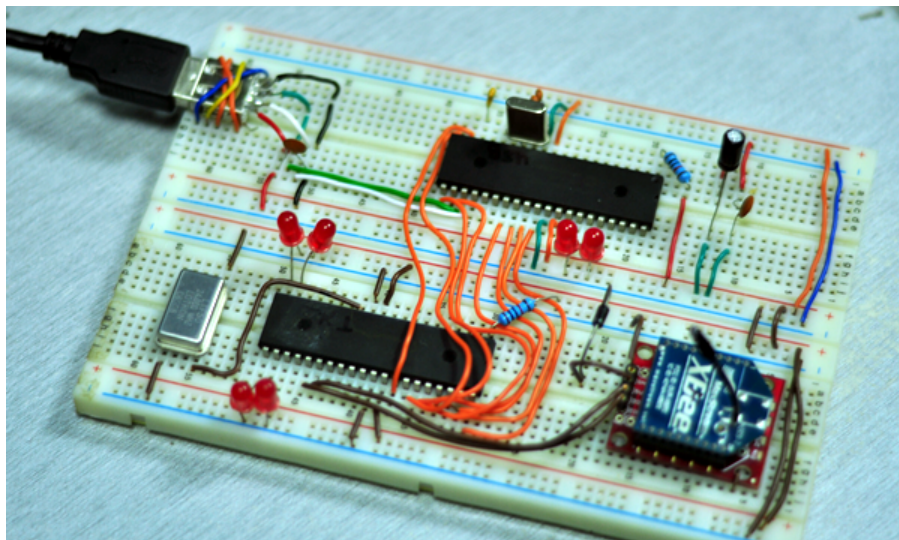


Figure 16: USB device constructed on breadboard

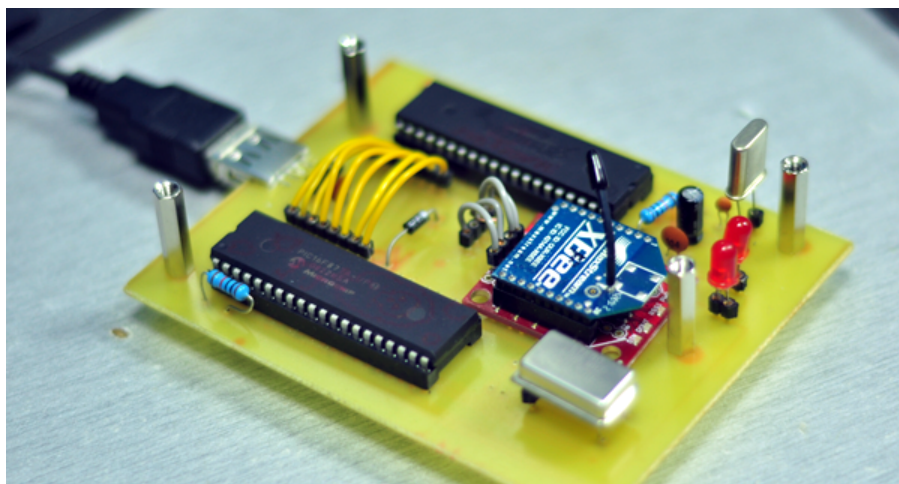


Figure 17: USB device constructed on custom PCB

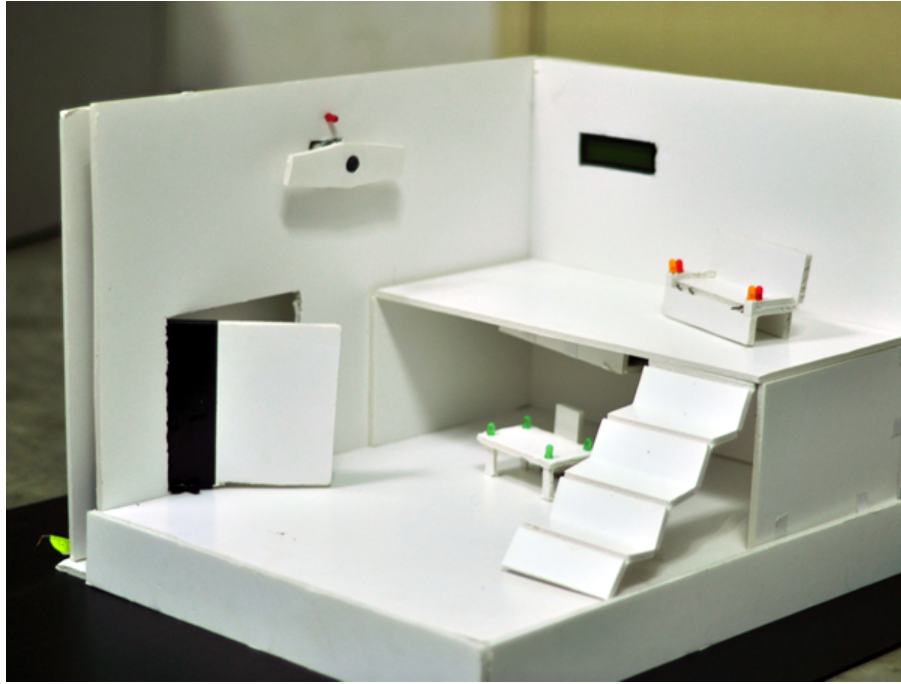


Figure 18: Small-scale model of the house

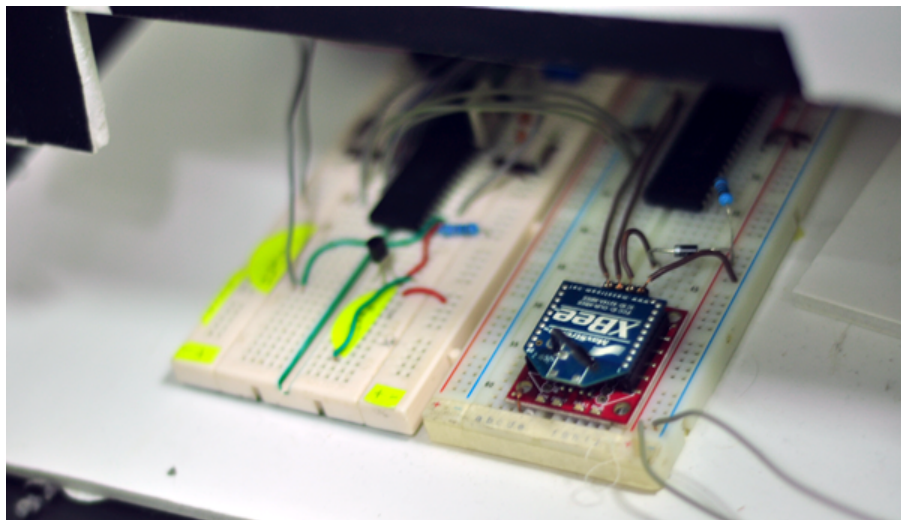


Figure 19: Receiver modules and other home devices circuits

Note that as of this writing, the USB device constructed on the custom printed circuit board (PCB) in Figure 17 is having some stability issue on the wireless transmission portion.

## 4.2 Usage Requirements

In order to use try or use the whole system, there are several requirements that has to be met first. The main requirements are:

Table 11: Usage requirements of the system

No	Item	Description
1	Host computer with Internet connection	The host computer has to be ON at all time (unless the user decided to turn the system OFF) and must be equipped with a reliable Internet connection
2	Status fetcher and XML parser	This locally hosted web-based page has to be open at all time in order to continuously fetching and parsing the latest status update from the user's Twitter account
3	Custom Windows application	This application has to be running at all time to periodically send the instruction to USB device
4	USB Device	This hardware needs to be connected to the host computer at all time to receive latest instruction and to transmit it wirelessly. No external power supply required.
5	Receiver and home devices	The receiver and devices must be connected, and can be place anywhere within the range, to be controlled by the user through the system.



### 4.3 Full System Flow and Integration

There are several steps involve in the full system of the working prototype. The user triggers the action, and the rest is handled automatically by the system, assuming the usage requirements are met (refer Table 11). In this section, a simple scenario will be illustrated to help readers in fully understand how the whole system work, and how the instruction propagated through the integrated components.

#### Step 1: Sending Instruction via Twitter

Twitter is a very accessible and portable to the user. Instruction can be sent at anytime and anywhere via several methods that users are familiar with. Assuming an instruction to turn ON the lights is sent to Twitter using any of the available mediums.

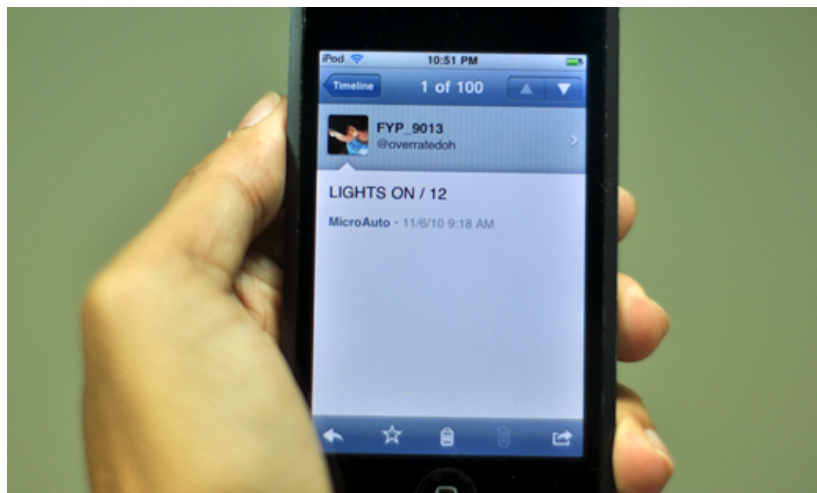


Figure 20: Accessing Twitter using mobile application

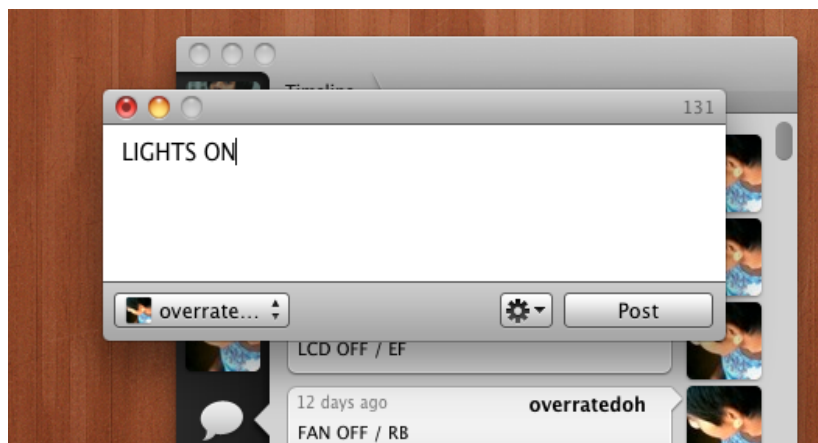


Figure 21: Accessing Twitter using desktop application



## **Step 2: Fetching status and parsing XML to a text file**

The status fetcher and XML parser will continuously get the latest status update from Twitter (refer **Section 3.4**). Therefore, this web-based page will have to be running locally all the time in the host computer. It will read the latest status update in XML formatted file, parse the unnecessary information, and store it inside a text file. In this case, based on Step 1, the text file will contain the phrase LIGHTS ON.

## **Step 3: Feeding instruction using custom Windows application**

The custom Windows application will continuously read the content of the text file (refer **Section 3.5.4**), produced in Step 2. In this case, the text file will contain the phrase “LIGHTS ON”. The application will also translate this phrase based on the pre-defined actions (refer **Section 3.5.3**) and a value of “5” will be fed to the USB device via USB connectivity (refer to Table 6 ).

## **Step 4: Decoding instruction on USB device**

Once the USB device receive the value fed in Step 3, it will decode it and translate it to a 8-bit binary number which has been defined inside the firmware. This binary number is the output of the first microcontroller. In this case, the value 5 fed by the Windows application will, in turn, produce an output 00000101 to PORT D of the microcontroller (refer Table 6).

## **Step 5: Transmitting character from USB device**

The output of the first microcontroller will be the input of the microcontroller of the transmitter circuit. This input is fed into the PORT B of the microcontroller. In this wireless module, instead of transmitting the binary number, it will be transmitting one single unique character, which has been assigned to every binary number (refer to Table 6). In this case, the corresponding character that is to be transmitted by XBee - based on the previous binary input - is the character “e”.

## **Step 6: Receiving transmitted characters**

The unique character that has been transmitted by the transmitter on USB device will be received on the receiver end. This character will be translated by microcontroller on the receiver end, to turn ON or OFF the selected devices, which has been connected to the receiver circuit. In this case, the character “e” received previously will trigger Pin B2, which in the small-scale house model, is connected to the lights – thus turning them ON.

The six steps above show how the instruction sent by the user via Twitter is propagated through the system. Several translating and decoding involve throughout the system in order to keep each instruction unique to only one single device. This is to avoid any conflicts and issues that might effects other devices.

## **4.4 USB Device Failure Analysis**

During the testing stage, the USB device exhibits a very small instability issues when it is connected to the host computer. This instability is observed on the two indicator lights, which supposed to be alternately blinking on normal condition. However, the percentage of the issues is very small, and the condition cannot be manually reproduced.

Therefore, a stress test is conducted on the USB device to measure its failure rate. This stress test is conducted by leaving the device running for 2 hours. Note that the test doesn't cover the whole system, especially the home devices automation. Steps involved in procedures of the test are:

1. Open custom Windows application on host computer
2. Connect USB device to host computer
3. Observe any undesired condition on USB device – especially the indicator lights – for a complete 2 hours.
4. Change the interval time of the custom Windows application
5. Repeat test for next iterations

Table 12: Stress test with 1 second timer interval

Iteration	Duration completed	Observation	Remarks
1	2 hours	No issues	The USB device is running smoothly throughout the test
2	2 hours	No issues	
3	2 hours	No issues	

Table 13: Stress test with 10 seconds timer interval

Iteration	Duration completed	Observation	Remarks
1	2 hours	No issues	The USB device is running smoothly throughout the test
2	2 hours	No issues	
3	2 hours	No issues	

Table 14: Stress test with 30 seconds timer interval

Iteration	Duration completed	Observation	Remarks
1	2 hours	No issues	The USB device is running smoothly throughout the test
2	2 hours	No issues	
3	2 hours	No issues	

Table 12 to 14 shows the result of the stress test. Note that the timer interval in the application is used as the manipulative variable in this stress test to see if there is any effect on the instability of the USB device. Based on the result, all 9 iterations produced 100% reliability on the system with the entire test period passed. The USB device did not failed any of the 2 hours stress tests iterations.

## 4.5 Optimal Fetch and Feed Intervals

In this working prototype, there are two different time interval or timer that is used inside the system. The first timer is used inside the status fetcher and XML parser, where the latest status update is continuously fetched from Twitter in a specified interval. The interval is defined on top of the source file, using header refresh method (refer to **Appendix D**).

The second timer is used inside the custom Windows application, where it is used to periodically read the content of a text file. It is defined on the top part of source file *MicroAuto.cs* (refer **Section 3.5.3** and **Appendix E**).

These two timers do not necessarily have the same interval value (in seconds), as they are independent from each other. They can be set to any value possible, down to 0 second. However, finding an optimal value for these interval is important in order not to burden the whole system, and at the same time not to compromise on the almost real-time experience, as expected by the users.

## **CHAPTER 5**

### **CONCLUSION AND RECOMMENDATIONS**

Throughout the research and exploration stage of the project, we have looked into several approaches and alternatives in order to choose the right methods, techniques and technologies for the project. During the development stage, all of the outcomes from each component are tested and integrated into one fully working prototype.

#### **5.1 Conclusion**

The approach used in this project - by dividing the whole project into four major components - is essential as it allows for parallel development process. Since most of them are not really dependent on each other, the development of each component can be done separately at the same time. This definitely eases the project management as well as speeding up the development processes. At the end of development stage, all the components are integrated.

The outcome of this project – which is mainly the physical prototype – shows that the objective and goals of this project has been successfully executed and all the requirements have been met. Starting out as a proof-of-concept idea, this project manages to illustrate the ability of Internet application/service to be connected to the real world environment. Although the project mainly focusing on Twitter to represent “the Internet application”, and home automation to represent “real world environment”, it is believed that these two variables may be change to other suitable options.

## 5.2 Recommendations

This project has a huge potential to be improved further if the right amount of time and resources is allocated. Although the current concept is working as intended, there are several recommendations that can be considered in order to enhance and improve the project to obtain much better outcome in the user point of view.

1. Consider two-way communication, which enables the home devices to update their current status, by the same mean of updating the Twitter status or sending a reply/ message to the user. This will allow a better interactivity throughout the system.
2. Make a conversion from the custom Windows application to a cross-platform desktop application – that can be used in any operating system - for a seamless user experience. One of the technologies that can be explored is Adobe AIR application platform as it is independent of the operating system.
3. Skip the web-based page for the status fetcher and XML parser. These functionalities should be able to be integrated directly into the custom Windows application. With this, it will further reduce the complexity and steps required for the data propagation through the system.
4. An advanced enhancement can be done by removing the dependency towards the host computer, but develop an embedded device and application with Internet capabilities that is connected directly to the microcontroller system instead. Open source operating system, or Windows Embedded CE can be used as the backbone of the system. However extensive knowledge in developing the device drivers for these operating systems is required.

There are a lot more improvement and enhancement that could be included in this project. With an increasing number of Internet users, and as the popularity of Twitter as a social networking website increases, this project will be proven to be a solid idea for an alternative for simpler home automation system.

## REFERENCES

- [1] TweetMyMac, .[Online]. Available: <http://thetacbox.co.uk/tweetmymac/> [Accessed 18 March 2010]
- [2] Botanicalls, “Botanicalls Twitter DIY” February 2008. [Online]. Available: [http://www.botanicalls.com/archived\\_kits/twitter/](http://www.botanicalls.com/archived_kits/twitter/) [Accessed: 18 March 2010]
- [3] Ruben Posada-Gomez, Jose Jorge Enriquez-Rodriguez, Giner Alor-Hernandez and Albino Martinez-Sibaja, “USB bulk transfers between a PC and a PIC microcontroller for embedded applications”, *Electronics, Robotics and Automotive Mechanics Conference*, 2008
- [4] Biz Stone, “What's The Deal with OAuth?”, 22 April 2009. [Online]. Available: <http://bit.ly/oGZ2f> [Accessed: 18 March 2010].
- [5] Alison Gianotto, “Writing Your First Twitter Application With OAuth”, 23 July 2009. [Online]. Available: <http://bit.ly/SPgXK> [Accessed: 18 March 2010]
- [5] Total Phase Knowledge Base - Article 10048, “USB vs. Serial and Parallel”. [Online]. Available: <http://bit.ly/b94jJn> [Accessed: 18 March 2010]
- [6] Simon Inns, “Building a PIC18F USB device”, 1 April 2010. [Online]. Available: <http://bit.ly/aApzbU> [Accessed: 23 April 2010]
- [7] Microchip Technology Inc, “PIC18F2455/2550/4455/4550 Data Sheet”, 2009
- [8] Maxstream, “XBee<sup>TM</sup>/XBee-PRO<sup>TM</sup> OEM RF Modules Manual”, 2006

## **APPENDICES**



## APPENDIX A

### Source Code for Web-based Interface Application

These are the essential source codes developed for the project. Other required files by third-party developers, such as the library files for OAuth authentication is included in the CD-ROM attached.

File name : index.php

Description : The main page before the user logged in using their Twitter account

```
<?php
session_start();
include("header.php"); ?>

<div class="front">
<?php /* Destroy the session if the user is logging out */
if ((isset($_GET['logout'])) && ($_GET['logout']=='true')) {
    echo 'LOGOUT';
    session_destroy();
    session_unset();
}

if ( isset($_SESSION['loggedin']) ||
isset($_SESSION['oauth_access_token'])) { ;?>

    <p>You're logged in<p>
    <p><a href="callback.php">Go to dashboard!</a></p>
    <p><a href="index.php?logout=true">Logout</a></p>

<?php
} else {
/* Include the config file */
require_once('config.php');

/* include the twitter OAuth library files */
require_once('oauth/twitterOAuth.php');
require_once('oauth/OAuth.php');

/*
Create a new TwitterOAuth object, and then get a request token. The
request
token will be used to build the link the user will use to authorize
the
```

```
application.
```

You should probably use a try/catch here to handle errors gracefully

```
*/
$to = new TwitterOAuth($consumer_key, $consumer_secret);
$tok = $to->getRequestToken();

$request_link = $to->getAuthorizeURL($tok);

/*
Save tokens for later - we need these on the callback page to ask
for the
access tokens
*/
$_SESSION['oauth_request_token'] = $token = $tok['oauth_token'];
$_SESSION['oauth_request_token_secret'] =
$tok['oauth_token_secret'];

echo '<p><a class="login" href="'. $request_link. '>Login via
Twitter</a>';

} ;?>
</div><!end of front>
<?php include("footer.php"); ?>
```

File name : callback.php

Description : Display panel page where the buttons can be seen once user logged-in via the OAuth authentication and received their access token.

```
<?php
session_start();
include("header.php");
include("actions.php");

/* Include the config file */
require_once('config.php');

/* include the twitter OAuth library files */
```

```

require_once('oauth/twitterOAuth.php');
require_once('oauth/OAuth.php');

/* check for an auth access token. If there's no auth token set, go
ahead and fetch one from Twitter,
* using the API call. */
if ((!isset($_SESSION['oauth_access_token'])) ||
($_SESSION['oauth_access_token']=='')) {

    $to = new TwitterOAuth($consumer_key, $consumer_secret,
$_SESSION['oauth_request_token'],
$_SESSION['oauth_request_token_secret']);
    $tok = $to->getAccessToken();

    /* Save tokens for later - might be wise to
    * store the oauth_token and secret in a database, and
    * only store the oauth_token in a cookie or session for
security purposes */
    $_SESSION['oauth_access_token'] = $token =
$tok['oauth_token'];
    $_SESSION['oauth_access_token_secret'] =
$tok['oauth_token_secret'];

}

/* Connect to the Twitter API */
$to = new TwitterOAuth($consumer_key, $consumer_secret,
$_SESSION['oauth_access_token'],
$_SESSION['oauth_access_token_secret']);
$content = $to-
>OAuthRequest('http://api.twitter.com/1/account/verify_credentials.x
ml', array(), 'GET');
$user = simplexml_load_string($content);

if ($user->screen_name!='') {
    $_SESSION['loggedin'] = 1;

    ?>
    <div class="greetbox">

        <a href="http://www.twitter.com/<?php echo $user->screen_name
;?>"> </a>

    <div class="greetmid">
        <p><strong>Hello,</strong></p>
        <p><h2><?php echo $user->screen_name ;?></h2></p>

```

```

</div>

<div class="greetout">

    <p><a class="login"
href="index.php?logout=true">Logout</a></p>
    </div>

<div class="clear"></div>

</div>

<?php // $action_1 = str_replace('+',' ', $action1); ?>
    <div id="actions">

        <ul>
            <li><a href="status.php?action=<?php echo $action1
;?>&placeValuesBeforeTB_=savedValues&TB_iframe=true&height=140&width
=200&modal=true" title="Sending instruction" class="thickbox"><?php
echo $action1 ;?></a></li>

            <li><a href="status.php?action=<?php echo $action2
;?>&placeValuesBeforeTB_=savedValues&TB_iframe=true&height=140&width
=200&modal=true" title="Sending instruction" class="thickbox"><?php
echo $action2 ;?></a></li>

            <li><a href="status.php?action=<?php echo $action3
;?>&placeValuesBeforeTB_=savedValues&TB_iframe=true&height=140&width
=200&modal=true" title="Sending instruction" class="thickbox"><?php
echo $action3 ;?></a></li>

            <li><a href="status.php?action=<?php echo $action4
;?>&placeValuesBeforeTB_=savedValues&TB_iframe=true&height=140&width
=200&modal=true" title="Sending instruction" class="thickbox"><?php
echo $action4 ;?></a></li>

            <li><a href="status.php?action=<?php echo $action5
;?>&placeValuesBeforeTB_=savedValues&TB_iframe=true&height=140&width
=200&modal=true" title="Sending instruction" class="thickbox"><?php
echo $action5 ;?></a></li>

            <li><a href="status.php?action=<?php echo $action6
;?>&placeValuesBeforeTB_=savedValues&TB_iframe=true&height=140&width
=200&modal=true" title="Sending instruction" class="thickbox"><?php
echo $action6 ;?></a></li>

            <li><a href="status.php?action=<?php echo $action7
;?>&placeValuesBeforeTB_=savedValues&TB_iframe=true&height=140&width
=200&modal=true" title="Sending instruction" class="thickbox"><?php
echo $action7 ;?></a></li>

            <li><a href="status.php?action=<?php echo $action8
;?>&placeValuesBeforeTB_=savedValues&TB_iframe=true&height=140&width
=200&modal=true" title="Sending instruction" class="thickbox"><?php
echo $action8 ;?></a></li>

            <li><a href="status.php?action=<?php echo $action9

```

```

;?>&placeValuesBeforeTB_=savedValues&TB_iframe=true&height=140&width
=200&modal=true" title="Sending instruction" class="thickbox"><?php
echo $action9 ;?></a></li>

    <li><a href="status.php?action=<?php echo $action10
;?>&placeValuesBeforeTB_=savedValues&TB_iframe=true&height=140&width
=200&modal=true" title="Sending instruction" class="thickbox"><?php
echo $action10 ;?></a></li>

    </ul>

</div>

<div id="recent">
<h3>Recently sent</h3>
<?php
$x = $to-
>OAuthRequest('http://api.twitter.com/1/statuses/user_timeline.xml',
array("count" => 5), 'GET');
$user = simplexml_load_string($x);
echo '<ul>';
foreach($user->status as $status){
echo '<li>'.$status->text;
echo '<br /><small>'.$status->created_at.'</small></li>';
}
echo '</ul>';
?>
</div>
<div class="clear"></div>
</div>
<?php

} else {
    echo 'Oops - an error has occurred.';
}

include("footer.php");
?>

```

File name : status.php

Description : An intermediate page to send status update to Twitter once the user clicked on the button.

```
<?php session_start();?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>MicroAuto - a Final Year Project</title>
<style type="text/css">
.modalsent{
background: lightblue;
font-family: "Helvetica Neue", Verdana;
text-align: center;
font-size: 12px;
}

p.actionsent{
padding: 5px 10px;
background: #333;
font-size: 13px;
color: #fff;
}
</style>
</head>

<body>

<div class="modalsent">
  <?php
/* Include the config file */
require_once('config.php');

/* include the twitter OAuth library files */
require_once('oauth/twitterOAuth.php');
require_once('oauth/OAuth.php');

/* check for an auth access token. If there's no auth token set, go
ahead and fetch one from Twitter,
* using the API call. */
if ((!isset($_SESSION['oauth_access_token'])) ||
($_SESSION['oauth_access_token']=='') {
```

```

        $to = new TwitterOAuth($consumer_key, $consumer_secret,
$_SESSION['oauth_request_token'],
$_SESSION['oauth_request_token_secret']);
        $tok = $to->getAccessToken();

        /* Save tokens for later - might be wise to
        * store the oauth_token and secret in a database, and
        * only store the oauth_token in a cookie or session for
        security purposes */
        $_SESSION['oauth_access_token'] = $token =
$tok['oauth_token'];
        $_SESSION['oauth_access_token_secret'] =
$tok['oauth_token_secret'];

    }

    /* Connect to the Twitter API */
    $to = new TwitterOAuth($consumer_key, $consumer_secret,
$_SESSION['oauth_access_token'],
$_SESSION['oauth_access_token_secret']);
    $content = $to-
>OAuthRequest('https://twitter.com/account/verify_credentials.xml',
array(), 'GET');
    $user = simplexml_load_string($content);

    if ($user->screen_name!='') {

        $updet=$_GET['action'];
        $updetrand=$updet." / ".rand(10,99);
        $content = simplexml_load_string($to-
>OAuthRequest('https://twitter.com/statuses/update.xml',
array('status' => $updetrand), 'POST'));

        if ($content){
            echo "<p>The following action has been sent: </p>";
            echo "<p class='actionsent'>".$updet."</p> ";
            //echo "<p><a
href='callback.php?oauth_token=".$_SESSION['oauth_request_token']."'
>Go to dashboard!</a></p>";
            ?>
            <p><input type="submit" id="Login" value="Close"
onclick="self.parent.tb_remove();" /></p>
            <?php
            /*<meta http-equiv="refresh" content="1;
URL=callback.php?oauth_token=<?php echo
$_SESSION['oauth_request_token'];?>">*/

        }else{

```

```
        echo "An error has occurred. Please try again.";
    }

} else {
    echo 'Oops - an error has occurred.';
}

?>
</div>

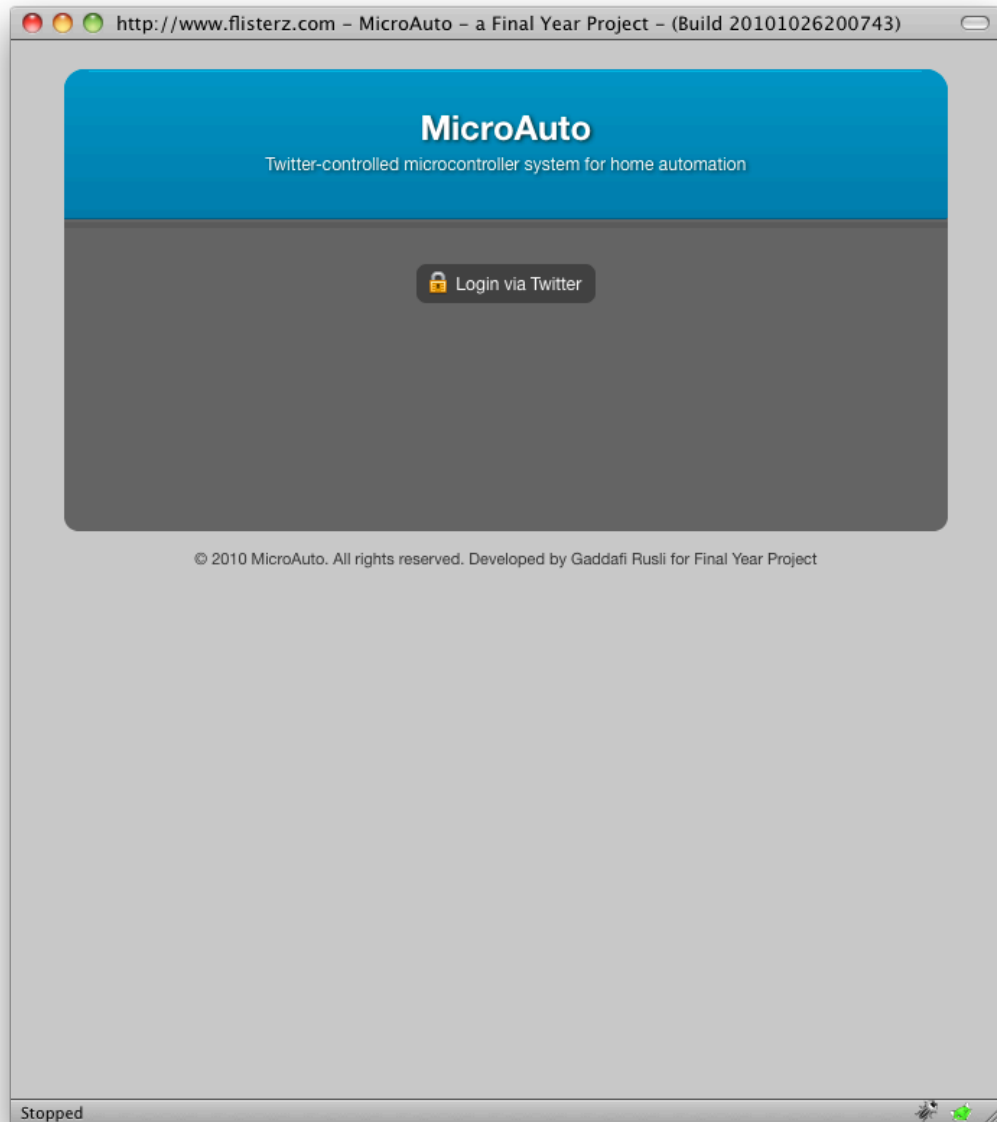
</body>
</html>
```



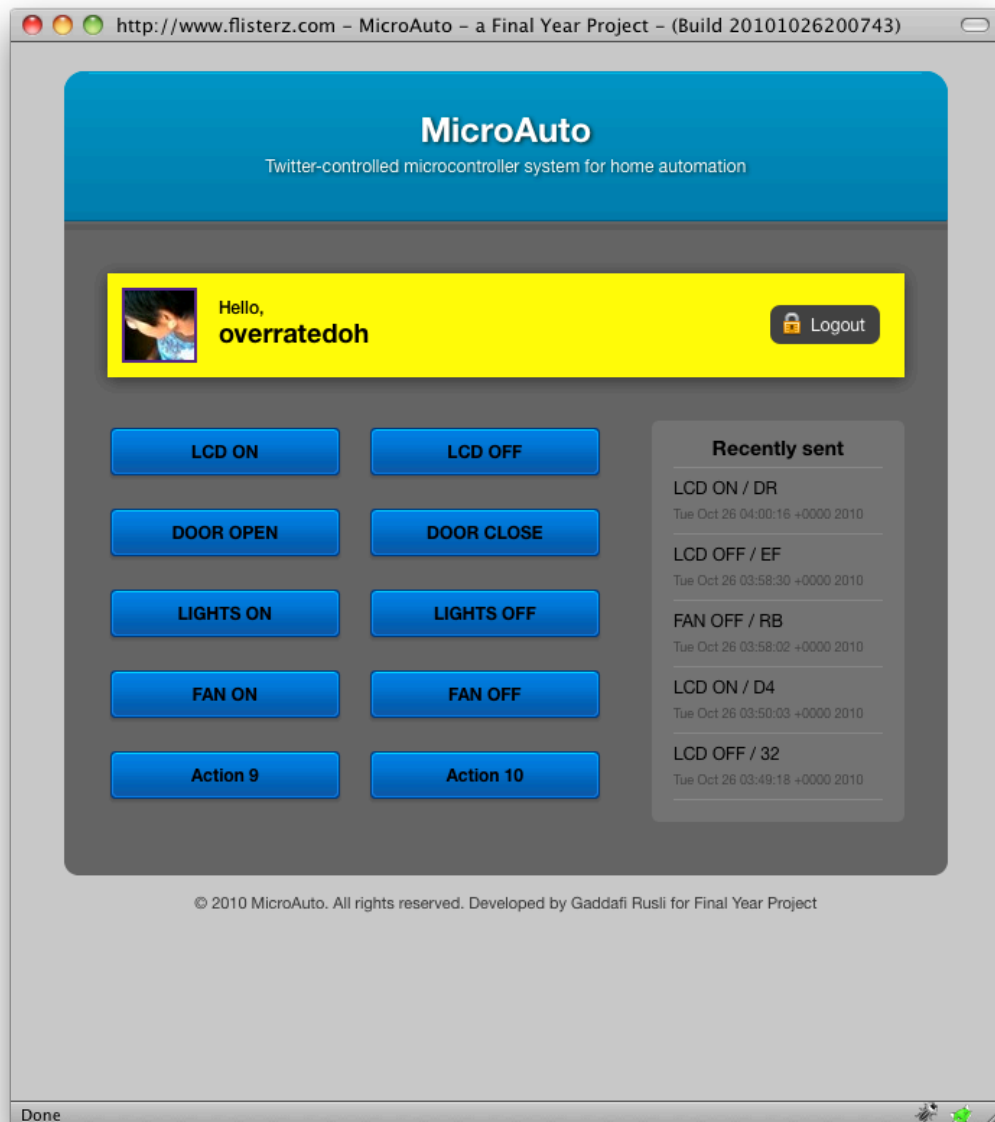
## APPENDIX B

### Screenshots of the Web-based Application

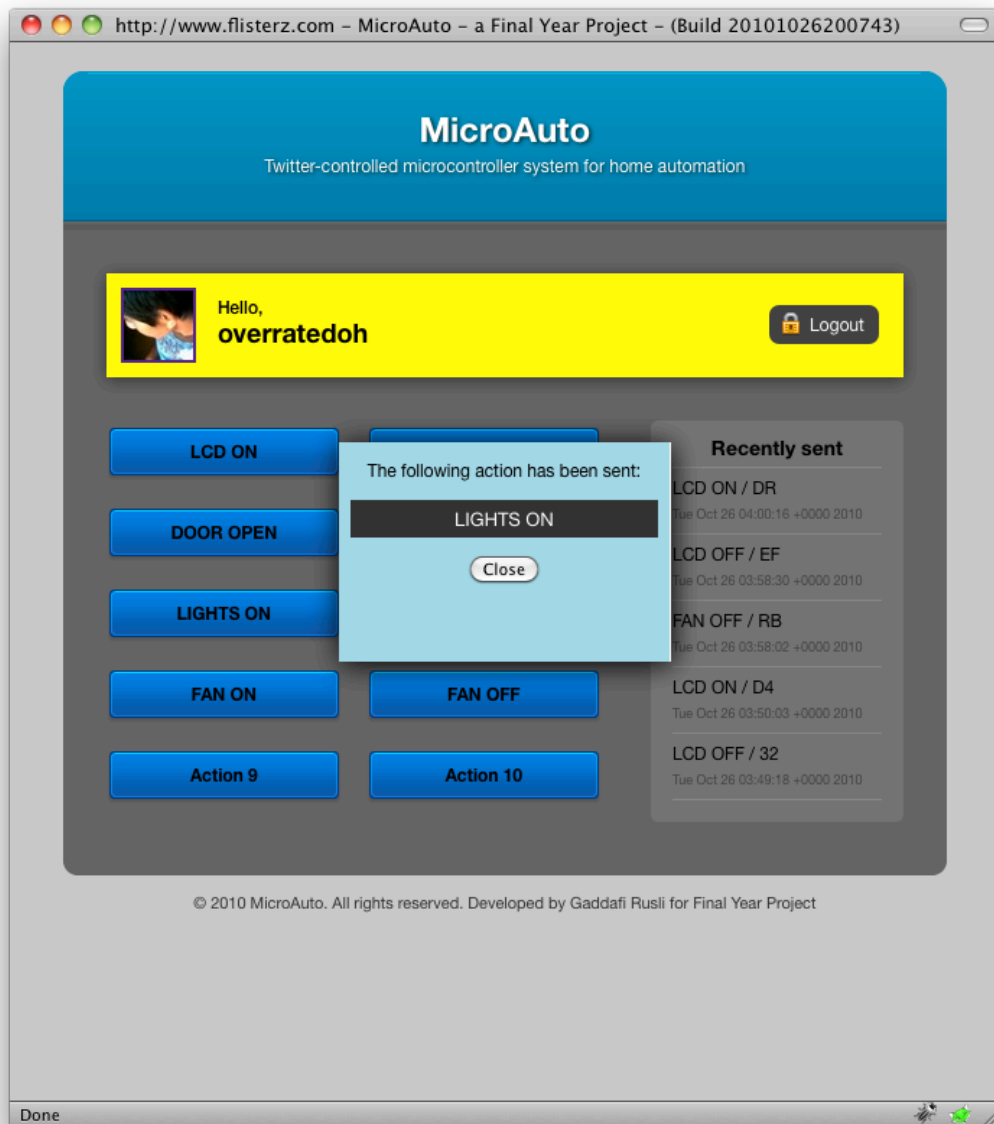
These are some of the screenshots taken on the web-based application interface, developed for this particular project.



The main page before the user login using their Twitter account



The panel page once the user logged in, showing the buttons representing the pre-defined actions that can be performed on the home devices.



A modal box appeared on the top layer of the page for confirmation that the instruction has been successfully sent.

## APPENDIX C

### XML-formatted Status Information

This screenshot shows data for a single status update from Twitter

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <statuses type="array">
- <status>
  <created_at>Sun Sep 12 13:36:39 +0000 2010</created_at>
  <id>24283872099</id>
  <text>Action#4 / 89</text>
  <source>web</source>
  <truncated>>false</truncated>
  <in_reply_to_status_id/>
  <in_reply_to_user_id/>
  <favorited>>false</favorited>
  <in_reply_to_screen_name/>
  <retweet_count/>
  <retweeted>>false</retweeted>
- <user>
  <id>92823647</id>
  <name>overrated</name>
  <screen_name>overratedoh</screen_name>
  <location/>
  <description/>
- <profile_image_url>
  http://a3.twimg.com/profile_images/972851479/094d9ff0f443412d91048379bcbb6131.jpeg_normal.jpg
  </profile_image_url>
  <url/>
  <protected>>false</protected>
  <followers_count>1</followers_count>
  <profile_background_color>C0DEED</profile_background_color>
  <profile_text_color>333333</profile_text_color>
  <profile_link_color>0084B4</profile_link_color>
  <profile_sidebar_fill_color>DDEEF6</profile_sidebar_fill_color>
  <profile_sidebar_border_color>C0DEED</profile_sidebar_border_color>
  <friends_count>0</friends_count>
  <created_at>Thu Nov 26 19:56:10 +0000 2009</created_at>
  <favourites_count>0</favourites_count>
  <utc_offset/>
  <time_zone/>
- <profile_background_image_url>
  http://s.twimg.com/a/1287010001/images/themes/theme1/bg.png
  </profile_background_image_url>
  <profile_background_tile>>false</profile_background_tile>
  <profile_use_background_image>true</profile_use_background_image>
  <notifications/>
  <geo_enabled>>false</geo_enabled>
  <verified>>false</verified>
  <following/>
  <statuses_count>121</statuses_count>
  <lang>en</lang>
  <contributors_enabled>>false</contributors_enabled>
  <follow_request_sent/>
  <listed_count>0</listed_count>
  <show_all_inline_media>>false</show_all_inline_media>
</user>
<geo/>
<coordinates/>
<place/>
<contributors/>
</status>
</statuses>
```

## APPENDIX D

### Source Code for Status Fetcher and XML Parser

File name : index.php

Description : This file fetches statuses from a user in XML format and parse the XML file to a readable string, and store it into a text file.

```
<?php
header("Refresh: 25;");

$username = "overratedoh";
$source =
"http://twitter.com/statuses/user_timeline/".$username.".xml?count=1
";

$xml = simplexml_load_file($source);
$raw = $xml->status->text;
if (strpos($raw, '/') !== false) {
    $content = "<span>".$raw."</span>";
    $actBeg = strpos($content, '<span>', 0);
    $actMid = substr($content, $actBeg+6);
    $actEnd = strpos($actMid, ' /');
    $action = substr($post, 0, $actEnd);
}else{
    $action = $raw;
}
echo $action;
file_put_contents("action.txt",$action);
?>
```

## APPENDIX E

### Source Code for Custom Windows Application

File name : MicroAuto.cs

Description : The user interface and main functionalities of the application

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using usb_api;
using System.IO;
using System.Threading;

namespace WindowsApplication3
{
    public class MicroAuto : System.Windows.Forms.Form
    {
        //define the read interval - in seconds
        public int second = 1;

        // instruction/action definition
        public string action1 = "LCD ON";
        public string action2 = "LCD OFF";
        public string action3 = "DOOR OPEN";
        public string action4 = "DOOR CLOSE";
        public string action5 = "LIGHTS ON";
        public string action6 = "LIGHTS OFF";
        public string action7 = "FAN ON";
        public string action8 = "FAN OFF";
        public string action9 = "Action#9";
        public string action10 = "Action#10";

        private System.Windows.Forms.Timer timer1;
        usb_interface usb_int = new usb_interface();
        private GroupBox groupBox1;
        private Label ReadAction;
        private Label label1;
        private IContainer components;
        private Label lblSec;
```

```

private Label label2;
public int interval;

public MicroAuto()
{
    InitializeComponent();
}

#region Clean up any resources being used.
protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose(disposing);
}
#endregion

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    this.groupBox1 = new System.Windows.Forms.GroupBox();
    this.ReadAction = new System.Windows.Forms.Label();
    this.lblSec = new System.Windows.Forms.Label();
    this.label1 = new System.Windows.Forms.Label();
    this.timer1 = new
System.Windows.Forms.Timer(this.components);
    this.label2 = new System.Windows.Forms.Label();
    this.groupBox1.SuspendLayout();
    this.SuspendLayout();

    //

```

```

// groupBox1
//
this.groupBox1.Controls.Add(this.ReadAction);
this.groupBox1.Controls.Add(this.lblSec);
this.groupBox1.Location = new System.Drawing.Point(34, 94);
this.groupBox1.Name = "groupBox1";
this.groupBox1.Size = new System.Drawing.Size(328, 107);
this.groupBox1.TabIndex = 9;
this.groupBox1.TabStop = false;
//
// ReadAction
//
this.ReadAction.Font = new System.Drawing.Font("Microsoft
Sans Serif", 14.25F);
this.ReadAction.Location = new System.Drawing.Point(20, 26);
this.ReadAction.Name = "ReadAction";
this.ReadAction.Size = new System.Drawing.Size(285, 56);
this.ReadAction.TabIndex = 9;
this.ReadAction.Text = "Reading data...";
//
// lblSec
//
this.lblSec.AutoSize = true;
this.lblSec.Font = new System.Drawing.Font("Microsoft Sans
Serif", 8.25F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.lblSec.Location = new System.Drawing.Point(249, 82);
this.lblSec.Name = "lblSec";
this.lblSec.Size = new System.Drawing.Size(56, 13);
this.lblSec.TabIndex = 12;
this.lblSec.Text = "5 seconds";
this.lblSec.TextAlign =
System.Drawing.ContentAlignment.TopRight;
//
// label1
//
this.label1.AutoSize = true;
this.label1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 24.25F, System.Drawing.FontStyle.Bold);
this.label1.Location = new System.Drawing.Point(110, 21);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(173, 38);

this.label1.TabIndex = 10;

```



```

        this.label1.Text = "MicroAuto";
        //
        // timer1
        //
        this.timer1.Enabled = true;
        this.timer1.Interval = 1000;
        this.timer1.Tick += new
System.EventHandler(this.timer1_Tick);
        //
        // label2
        //
        this.label2.Font = new System.Drawing.Font("Microsoft Sans
Serif", 8.25F);
        this.label2.Location = new System.Drawing.Point(38, 64);
        this.label2.Name = "label2";
        this.label2.Size = new System.Drawing.Size(326, 27);
        this.label2.TabIndex = 16;
        this.label2.Text = "Twitter-controlled Microcontroller
System for Home Automation";
        //
        // MicroAuto
        //
        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
        this.ClientSize = new System.Drawing.Size(392, 213);
        this.Controls.Add(this.label2);
        this.Controls.Add(this.label1);
        this.Controls.Add(this.groupBox1);
        this.Name = "MicroAuto";
        this.Text = "MicroAuto - FYP Project";
        this.groupBox1.ResumeLayout(false);
        this.groupBox1.PerformLayout();
        this.ResumeLayout(false);
        this.PerformLayout();
    }
#endregion

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]

static void Main()

```

```

    {
        Application.Run(new MicroAuto());
    }

// Function to read the content of the text file,
// as stored by the status fetcher and XML parser

public void Read_file()
{
    StreamReader textFile = new StreamReader("action.txt");
    string fileread = textFile.ReadToEnd();

    if (fileread == action1)
    {
        ReadAction.Text = action1;
        usb_int.actionSend(1, true);
    }
    else if (fileread == action2)
    {
        ReadAction.Text = action2;
        usb_int.actionSend(2, true);
    }
    else if (fileread == action3)
    {
        ReadAction.Text = action3;
        usb_int.actionSend(3, true);
    }
    else if (fileread == action4)
    {
        ReadAction.Text = action4;
        usb_int.actionSend(4, true);
    }
    else if (fileread == action5)
    {
        ReadAction.Text = action5;
        usb_int.actionSend(5, true);
    }
    else if (fileread == action6)
    {
        ReadAction.Text = action6;
        usb_int.actionSend(6, true);
    }
}

```

```

else if (fileread == action7)
{
    ReadAction.Text = action7;
    usb_int.actionSend(7, true);
}
else if (fileread == action8)
{
    ReadAction.Text = action8;
    usb_int.actionSend(8, true);
}
else if (fileread == action9)
{
    ReadAction.Text = action9;
    usb_int.actionSend(9, true);
}
else if (fileread == action10)
{
    ReadAction.Text = action10;
    usb_int.actionSend(10, true);
}
else
{
    ReadAction.Text = "Error! " + fileread + " -
unidentified action";
}
textFile.Close();
}

private void timer1_Tick(object sender, EventArgs e)
{
    // Verify if the time didn't pass.
    // Else continue counting.
    if (interval < 1)
    {
        interval = second;
        Read_file();
    }
    else
    {
        interval -= 1;
    }

    // Display the current values of interval in

```

```

        // the corresponding fields.
        lblSec.Text = interval.ToString() + " seconds";
    }
}
}

```

File name : usb\_interfaces.cs

Description : For USB connectivity and interaction with the USB device

```

using System;
using System.IO;
using System.Runtime.InteropServices;
using PVOID = System.IntPtr;
using DWORD = System.UInt32;

namespace usb_api
{
    unsafe public class usb_interface
    {
        #region String Definitions of Pipes and VID_PID
        //string vid_pid_boot= "vid_04d8&pid_000b";    //
        Bootloader vid_pid ID
        string vid_pid_norm= "vid_04d8&pid_000c";

        string out_pipe= "\\MCHP_EP1"; // Define End Points
        string in_pipe= "\\MCHP_EP1";
        #endregion

        #region Imported DLL functions from mpusbapi.dll
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBGetDLLVersion();
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBGetDeviceCount(string
        pVID_PID);
        [DllImport("mpusbapi.dll")]
        private static extern void* _MPUSBOpen(DWORD
        instance,string pVID_PID,string pEP,DWORD dwDir,DWORD dwReserved);
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBRead(void*

```

```

handle,void* pData,DWORD dwLen,DWORD* pLength,DWORD dwMilliseconds);
    [DllImport("mpusbapi.dll")]
    private static extern DWORD _MPUSBWrite(void*
handle,void* pData,DWORD dwLen,DWORD* pLength,DWORD dwMilliseconds);
    [DllImport("mpusbapi.dll")]
    private static extern DWORD _MPUSBReadInt(void*
handle,DWORD dwLen,DWORD* pLength,DWORD dwMilliseconds);

    [DllImport("mpusbapi.dll")]
    private static extern bool _MPUSBClose(void* handle);
#endregion

void* myOutPipe;
void* myInPipe;
private void OpenPipes()
{
    DWORD selection=0; // Selects the device to
connect to, in this example it is assumed you will only have one
device per vid_pid connected.

    myOutPipe =
_MPUSBOpen(selection,vid_pid_norm,out_pipe,0,0);
    myInPipe =
_MPUSBOpen(selection,vid_pid_norm,in_pipe,1,0);
}

private void ClosePipes()
{
    _MPUSBClose(myOutPipe);
    _MPUSBClose(myInPipe);
}

private DWORD SendReceivePacket(byte* SendData, DWORD
SendLength, byte* ReceiveData, DWORD *ReceiveLength)
{
    uint SendDelay=1000;
    uint ReceiveDelay=1000;
    DWORD SentDataLength;
    DWORD ExpectedReceiveLength = *ReceiveLength;
    OpenPipes();
    if(_MPUSBWrite(myOutPipe,(void*)SendData,SendLength,&SentDataL
ength,SendDelay)==1)
    {
        if(_MPUSBRead(myInPipe,(void*)ReceiveData,
ExpectedReceiveLength,ReceiveLength,ReceiveDelay)==1)
        {

```

```

ExpectedReceiveLength)                if(*ReceiveLength ==
                                        {
                                        ClosePipes();
                                        return 1;    // Success!
                                        }
                                        else if(*ReceiveLength <
ExpectedReceiveLength)
                                        {
                                        ClosePipes();
                                        return 2;    // Partially
failed, incorrect receive length      }
                                        }
                                        }
                                        ClosePipes();
                                        return 0;    // Operation Failed
                                    }

    public DWORD GetDLLVersion()
    {
        return _MPUSBGetDLLVersion();
    }

    public DWORD GetDeviceCount(string Vid_Pid)
    {
        return _MPUSBGetDeviceCount(Vid_Pid);
    }

    public int actionSend(uint led, bool State)
    {
// The default demo firmware application has a defined application
// level protocol.
// To set the LED's, the host must send the UPDATE_LED
// command which is defined as 0x32, followed by the LED to update,
// then the state to change the LED to.
// i.e. <UPDATE_LED><0x01><0x01>
//
// Would activate LED 1
//
// The receive buffer size must be equal to or larger than the
// maximum
// endpoint size it is communicating with. In this case, it is set
// to 64 bytes.

```

```

        byte* send_buf=stackalloc byte[64];
        byte* receive_buf=stackalloc byte[64];
        DWORD RecvLength=3;
        send_buf[0] = 0x32;      //Command for LED Status
        send_buf[1] = (byte)led;
        send_buf[2] = (byte)(State?1:0);

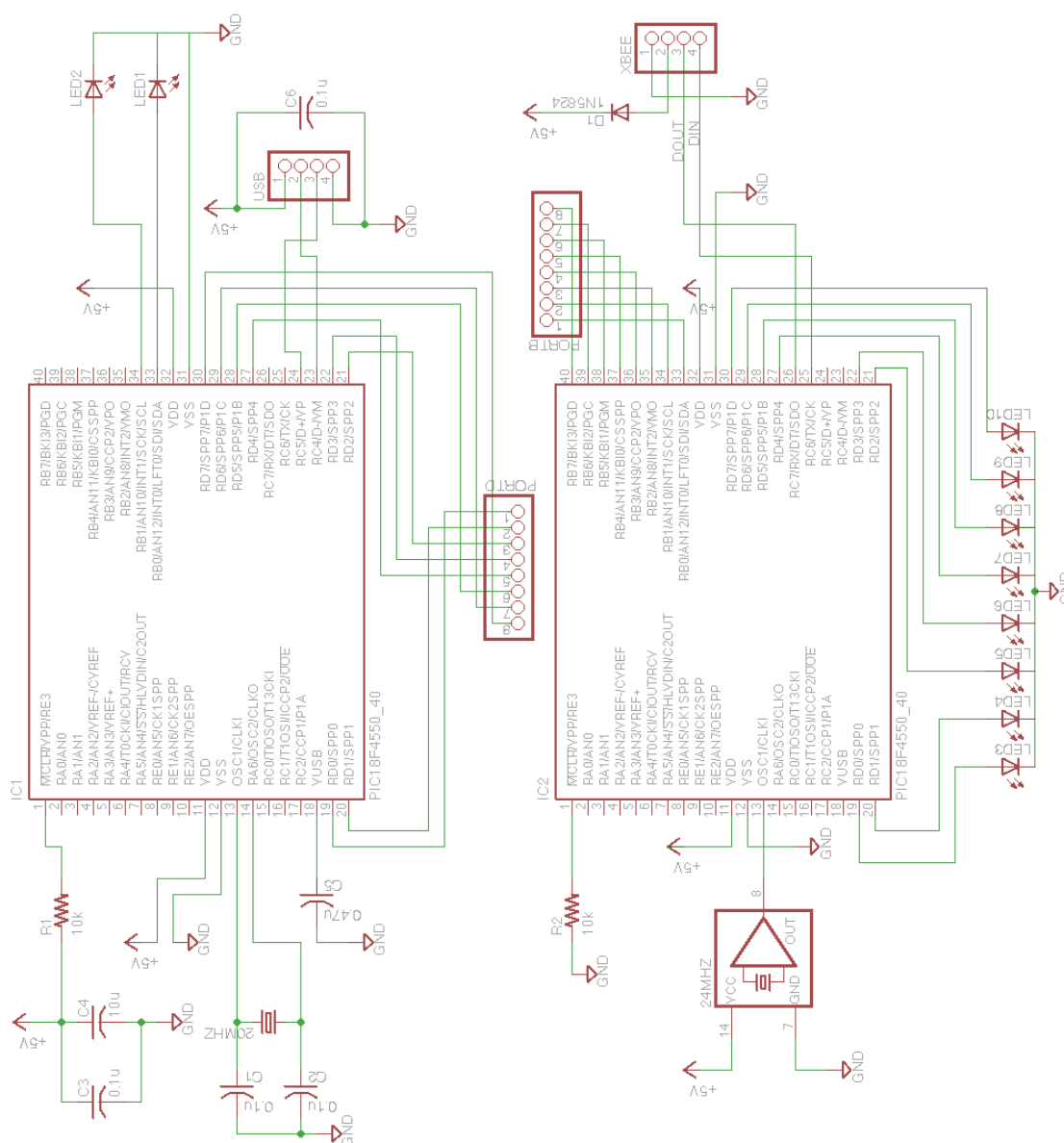
        if
(SendReceivePacket(send_buf,3,receive_buf,&RecvLength) == 1)
        {
            if (RecvLength == 1 && receive_buf[0] ==
0x32)
            {
                return 0;
            }
            else
            {
                return 2;
            }
        }
        else
        {
            return 1;
        }
    }
}

```

## APPENDIX F

## USB Device Schematic

The following figure shows the schematic diagram for the complete USB device. It consists of two part, which is the USB device itself, as well as the XBee transmitter circuit. These two circuits are only connected via PORT D to PORT B, for data and signal transfer between the two subsystems.

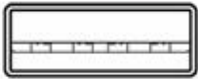



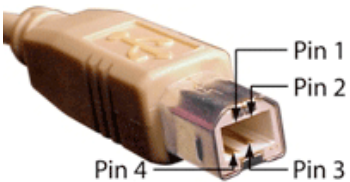
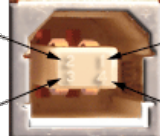



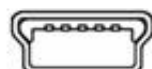
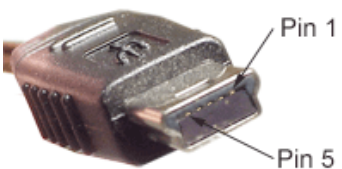





## APPENDIX G

### USB Connectors and Ports

The following table shows four types of USB connectors and ports that are widely use for various purposes.

Type	Port Image	Connectors	Receptacles
Type A	4.5mm x 12.0mm 	 Pin 1 Pin 4	 Pin1 Pin 4
Type B	7.3mm x 8.5mm 	 Pin 1 Pin 2 Pin 3 Pin 4	 Pin 2 Pin 1 Pin 3 Pin 4
Mini-A	3.0mm x 6.8mm 	 Pin 1 Pin 5	 Pin 1 Pin 5
Mini-B	3.0mm x 6.8mm 	 Pin 1 Pin 5	 Pin 1 Pin 5

Source

<http://www.intel.com/support/motherboards/desktop/sb/CS-023466.htm>

<http://www.accesscomms.com.au/reference/usb.htm>

## APPENDIX H

### Source Code for USB Device Firmware

The firmware source code that are included in this appendix are the main files that are highly modified for the purposes of this project. The other source files required for the firmware are included in the CD-ROM attached.

File name : io\_cfg.h

Description : This header file taken from Microchip sample code has been modified to suits the input and output configuration needs of the project.

```
/*
 *
 *      Microchip USB C18 Firmware Version 1.0
 *
 */
*****
*
* FileName:      io_cfg.h
* Dependencies:  See INCLUDES section below
* Processor:     PIC18
* Compiler:      C18 2.30.01+
* Company:       Microchip Technology, Inc.
*
* Software License Agreement
*
* The software supplied herewith by Microchip Technology
Incorporated
* (the "Company") for its PICmicro® Microcontroller is intended and
* supplied to you, the Company's customer, for use solely and
* exclusively on Microchip PICmicro Microcontroller products. The
* software is owned by the Company and/or its supplier, and is
* protected under applicable copyright laws. All rights are
reserved.
* Any use in violation of the foregoing restrictions may subject
the
* user to criminal sanctions under applicable laws, as well as to
* civil liability for the breach of the terms and conditions of
this
* license.
*
* THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
* WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
* TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
* PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
* CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
*
* Author          Date          Comment
* ~~~~~
* Rawin Rojvanit   11/19/04      Original.
```

```

*****
/

#ifndef IO_CFG_H
#define IO_CFG_H

/** I N C L U D E S
*****/
#include "autofiles\usbcfg.h"

/** T R I S
*****/
#define INPUT_PIN          1
#define OUTPUT_PIN         0

/** U S B
*****/
#define tris_usb_bus_sense  TRISAbits.TRISA1    // Input

#if defined(USE_USB_BUS_SENSE_IO)
#define usb_bus_sense      PORTAbits.RA1
#else
#define usb_bus_sense      1
#endif

#define tris_self_power     TRISAbits.TRISA2    // Input

#if defined(USE_SELF_POWER_SENSE_IO)
#define self_power          PORTAbits.RA2
#else
#define self_power          1
#endif

/** L E D
*****/
#define mInitAllLEDs()      LATD &= 0x00; TRISD &= 0x00; LATB &=
0x00;    TRISB &= 0x00;

#define mLED_A              LATBbits.LATB0
#define mLED_B              LATBbits.LATB1

#define mLED_1              LATDbits.LATD0
#define mLED_2              LATDbits.LATD1
#define mLED_3              LATDbits.LATD2
#define mLED_4              LATDbits.LATD3
#define mLED_5              LATDbits.LATD4
#define mLED_6              LATDbits.LATD5
#define mLED_7              LATDbits.LATD6
#define mLED_8              LATDbits.LATD7

#define mLED_A_On()         mLED_A = 1;
#define mLED_B_On()         mLED_B = 1;
#define mLED_1_On()         mLED_1 = 1;
#define mLED_2_On()         mLED_2 = 1;
#define mLED_3_On()         mLED_3 = 1;
#define mLED_4_On()         mLED_4 = 1;
#define mLED_5_On()         mLED_5 = 1;
#define mLED_6_On()         mLED_6 = 1;
#define mLED_7_On()         mLED_7 = 1;
#define mLED_8_On()         mLED_8 = 1;

```

```

#define mLED_A_Off()      mLED_A = 0;
#define mLED_B_Off()      mLED_B = 0;
#define mLED_1_Off()      mLED_1 = 0;
#define mLED_2_Off()      mLED_2 = 0;
#define mLED_3_Off()      mLED_3 = 0;
#define mLED_4_Off()      mLED_4 = 0;
#define mLED_5_Off()      mLED_5 = 0;
#define mLED_6_Off()      mLED_6 = 0;
#define mLED_7_Off()      mLED_7 = 0;
#define mLED_8_Off()      mLED_8 = 0;

#define mLED_A_Toggle()    mLED_A = !mLED_A;
#define mLED_B_Toggle()    mLED_B = !mLED_B;
#define mLED_1_Toggle()    mLED_1 = !mLED_1;
#define mLED_2_Toggle()    mLED_2 = !mLED_2;
#define mLED_3_Toggle()    mLED_3 = !mLED_3;
#define mLED_4_Toggle()    mLED_4 = !mLED_4;
#define mLED_5_Toggle()    mLED_5 = !mLED_5;
#define mLED_6_Toggle()    mLED_6 = !mLED_6;
#define mLED_7_Toggle()    mLED_7 = !mLED_7;
#define mLED_8_Toggle()    mLED_8 = !mLED_8;

#endif //IO_CFG_H

```

File name : user.c

Description : Handling the decoding on value received from custom Windows application to produce the 8-bit binary digit. It also handle the blinking state of indicator lights.

```

/*****
**
*
*           Microchip USB C18 Firmware Version 1.0
*
*****/

*
* FileName:      user.c
* Dependencies:  See INCLUDES section below
* Processor:     PIC18
* Compiler:      C18 2.30.01+
* Company:       Microchip Technology, Inc.
*
* Software License Agreement
*
* The software supplied herewith by Microchip Technology
Incorporated
* (the "Company") for its PICmicro® Microcontroller is intended and
* supplied to you, the Company's customer, for use solely and
* exclusively on Microchip PICmicro Microcontroller products. The
* software is owned by the Company and/or its supplier, and is
* protected under applicable copyright laws. All rights are
reserved.
* Any use in violation of the foregoing restrictions may subject
the

```

```

* user to criminal sanctions under applicable laws, as well as to
* civil liability for the breach of the terms and conditions of
this
* license.
*
* THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
* WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
* TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
* PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
* CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
*
* Author                Date                Comment
* ~~~~~
~
* Rawin Rojvanit        11/19/04        Original.

*****
/

/** I N C L U D E S
*****/
#include <p18cxxx.h>
#include <usart.h>
#include "system\typedefs.h"

#include "system\usb\usb.h"

#include "io_cfg.h"           // I/O pin mapping
#include "user\user.h"

/** V A R I A B L E S
*****/
#pragma udata

byte counter;
byte trf_state;

DATA_PACKET dataPacket;

/** P R I V A T E   P R O T O T Y P E S
*****/

void BlinkUSBStatus(void);
void ServiceRequests(void);

/** D E C L A R A T I O N S
*****/
#pragma code
void UserInit(void)
{
    mInitAllLEDs();
} //end UserInit

/*****
* Function:          void ProcessIO(void)
*
* PreCondition:      None

```

```

*
* Input:          None
*
* Output:         None
*
* Side Effects:   None
*
* Overview:       This function is a place holder for other user
routines.
*                It is a mixture of both USB and non-USB tasks.
*
* Note:           None

*****
*****/
void ProcessIO(void)
{
    BlinkUSBStatus();
    // User Application USB tasks
    if((usb_device_state < CONFIGURED_STATE) || (UCONbits.SUSPND==1))
return;

    ServiceRequests();
}

//end ProcessIO

void ServiceRequests(void)
{
    byte index;

    if(USBGenRead((byte*)&dataPacket,sizeof(dataPacket)))
    {
        counter = 0;
        switch(dataPacket.CMD)
        {
            case READ_VERSION:
                //dataPacket._byte[1] is len
                dataPacket._byte[2] = MINOR_VERSION;
                dataPacket._byte[3] = MAJOR_VERSION;
                counter=0x04;
                break;

            case ID_BOARD:
                counter = 0x01;
                if(dataPacket.ID == 0) {
                    mLED_3_Off();mLED_4_Off();
                }else if(dataPacket.ID == 1) {
                    mLED_3_Off();mLED_4_On();
                } else if(dataPacket.ID == 2) {
                    mLED_3_On();mLED_4_Off();
                } else if(dataPacket.ID == 3) {
                    mLED_3_On();mLED_4_On();
                } else
                    counter = 0x00;
                break;

            case UPDATE_LED:

                if(dataPacket.led_num == 1) {

                    mLED_1 = 1; mLED_2 = 0; mLED_3 = 0; mLED_4 = 0;

```

```

        mLED_5 = 0; mLED_6 = 0; mLED_7 = 0; mLED_8 = 0;
        counter = 0x01;

    } else if(dataPacket.led_num == 2) {

        mLED_1 = 0; mLED_2 = 1; mLED_3 = 0; mLED_4 = 0;
        mLED_5 = 0; mLED_6 = 0; mLED_7 = 0; mLED_8 = 0;
        counter = 0x01;

    } else if(dataPacket.led_num == 3) {

        mLED_1 = 1; mLED_2 = 1; mLED_3 = 0; mLED_4 = 0;
        mLED_5 = 0; mLED_6 = 0; mLED_7 = 0; mLED_8 = 0;
        counter = 0x01;

    } else if(dataPacket.led_num == 4) {

        mLED_1 = 0; mLED_2 = 0; mLED_3 = 1; mLED_4 = 0;
        mLED_5 = 0; mLED_6 = 0; mLED_7 = 0; mLED_8 = 0;
        counter = 0x01;

    } else if(dataPacket.led_num == 5) {

        mLED_1 = 1; mLED_2 = 0; mLED_3 = 1; mLED_4 = 0;
        mLED_5 = 0; mLED_6 = 0; mLED_7 = 0; mLED_8 = 0;
        counter = 0x01;

    } else if(dataPacket.led_num == 6) {

        mLED_1 = 0; mLED_2 = 1; mLED_3 = 1; mLED_4 = 0;
        mLED_5 = 0; mLED_6 = 0; mLED_7 = 0; mLED_8 = 0;
        counter = 0x01;

    } else if(dataPacket.led_num == 7) {

        mLED_1 = 1; mLED_2 = 1; mLED_3 = 1; mLED_4 = 0;
        mLED_5 = 0; mLED_6 = 0; mLED_7 = 0; mLED_8 = 0;
        counter = 0x01;

    } else if(dataPacket.led_num == 8) {

        mLED_1 = 0; mLED_2 = 0; mLED_3 = 0; mLED_4 = 1;
        mLED_5 = 0; mLED_6 = 0; mLED_7 = 0; mLED_8 = 0;
        counter = 0x01;

    } else {

        mLED_1 = 1; mLED_2 = 1; mLED_3 = 1; mLED_4 = 1;
        mLED_5 = 1; mLED_6 = 1; mLED_7 = 1; mLED_8 = 1;
        counter = 0x01;
    }

    break;

case RESET:
    Reset();
    break;

default:
    break;
} //end switch()

```

```

        if(counter != 0)
        {
            if(!mUSBGenTxIsBusy())
                USBGenWrite((byte*)&dataPacket,counter);
        }//end if
    }//end if

} //end ServiceRequests

/*****
*****
* Function:          void BlinkUSBStatus(void)
*
* PreCondition:      None
*
* Input:             None
*
* Output:            None
*
* Side Effects:      None
*
* Overview:          BlinkUSBStatus turns on and off LEDs
corresponding to
*                   the USB device state.
*
* Note:              mLED macros can be found in io_cfg.h
*                   usb_device_state is declared in usbmmmap.c and is
modified
*                   in usbdrv.c, usbctrltrf.c, and usb9.c
*****
*****/
void BlinkUSBStatus(void)
{
    static word led_count=0;

    if(led_count == 0)led_count = 10000U;
    led_count--;

#define mLED_Both_Off()      {mLED_A_Off();mLED_B_Off();}
#define mLED_Both_On()      {mLED_A_On();mLED_B_On();}
#define mLED_Only_A_On()    {mLED_A_On();mLED_B_Off();}
#define mLED_Only_B_On()    {mLED_A_Off();mLED_B_On();}

    if(UCONbits.SUSPND == 1)
    {
        if(led_count==0)
        {
            mLED_A_Toggle();
            mLED_B = mLED_A;          // Both blink at the same time
        } //end if
    }
    else
    {
        if(usb_device_state == DETACHED_STATE)
        {
            mLED_Both_Off();

        }
        else if(usb_device_state == ATTACHED_STATE)
        {
            mLED_Both_On();

```

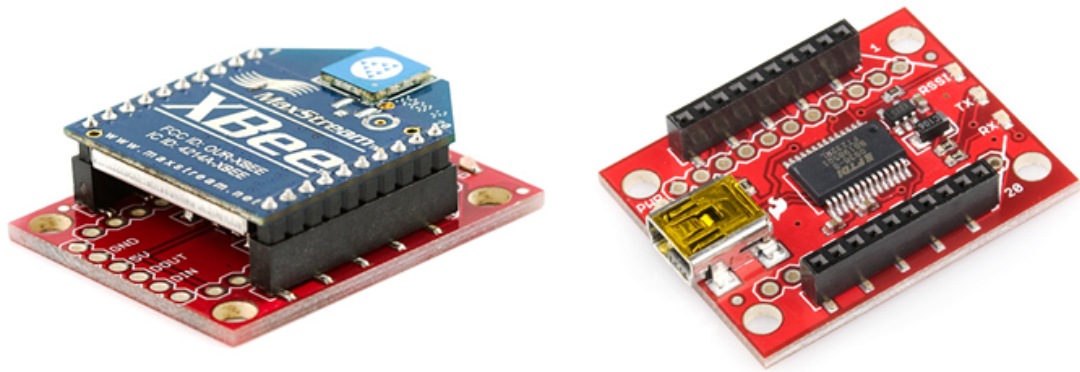




## APPENDIX I

### XBee Breakout Boards

The following breakout boards are used along with the usual XBee wireless module for easier circuit integration and configuration setting.



From left, XBee Explorer regulated and XBee Explorer USB

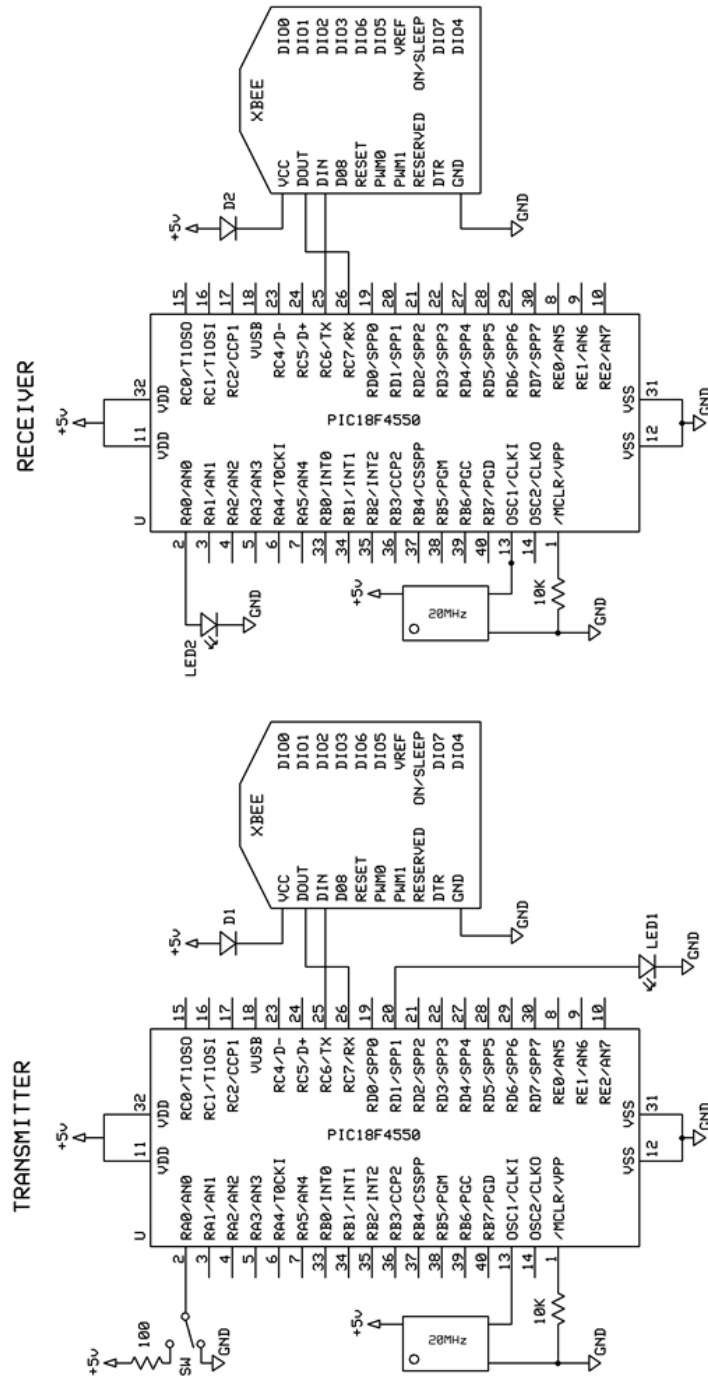
The XBee Explorer regulated is used to connect the XBee module to the microcontroller circuit. It has built-in voltage regulator, to regulate the voltage to 3.3V, which is the voltage used by the module. It also brings out the four most used pins to the front for easier access.

The XBee Explorer USB is almost the same as above, but it has additional USB connectivity capability to connect the module to host computer. This can be used to update firmware versions or configure the necessary setting using X-CTU application.

## APPENDIX J

### XBee Wireless Module Circuit Schematic

The following schematics are the basic schematics used to build the XBee transmitter circuit (which is combined into USB device – Appendix F) and XBee receiver circuit.



## APPENDIX K

### Source Code for Wireless Transmission

File name : transmitter.c

Description : This code is for the transmitter side, to get data from USB device, and transmit unique character accordingly.

```
#include <18f452.h>
#fuses HS,NOLVP,NOWDT,NOPROTECT
#use delay(clock=24000000)
#use rs232(baud=9600, UART1)
#include <stdlib.h>

void main() {
    while (TRUE) {
        if (input_b() == 0x01){
            output_d(0x01);
            printf("a");          //sends signal a
        }
        else if (input_b() == 0x02){
            output_d(0x02);
            printf("b");          //sends signal b
        }
        else if (input_b() == 0x03){
            output_d(0x03);
            printf("c");          //sends signal c
        }
        else if (input_b() == 0x04){
            output_d(0x04);
            printf("d");          //sends signal d
        }
        else if (input_b() == 0x05){
            output_d(0x05);
            printf("e");          //sends signal e
        }
        else if (input_b() == 0x06){
            output_d(0x06);
            printf("f");          //sends signal f
        }

        else if (input_b() == 0x07){
            output_d(0x07);
        }
    }
}
```

```

        printf("g");          //sends signal g
    }
    else if (input_b() == 0x08){
        output_d(0x08);
        printf("h");          //sends signal h
    }
    else if (input_b() == 0x09){
        output_d(0x09);
        printf("i");          //sends signal h
    }
    else if (input_b() == 0x0A){
        output_d(0x0A);
        printf("j");          //sends signal h
    }
    else if (input_b() == 0x00) {
        output_d(0x00);
        printf("x");          //sends signal x
    }
    else {
        output_d(0xFF);
        printf("x");          //sends signal x
    }
}
}

```

File name : receiver.c

Description : This code is for the receiver side, to receive character from transmitter, and decode it to control several devices.

```
#include <18f4550.h>
#fuses HS,NOLVP,NOWDT,NOPROTECT
#use delay(clock=24000000)
#use rs232(baud=9600, UART1)
#include <stdlib.h>
char x;

void main() {
output_b(0x00);
    while (TRUE) {
        if (kbhit()) {
            x = getc();
        }

        if (x=='a'){
            output_d(0x01);
            output_high(PIN_B0);
        }
        else if (x == 'b'){
            output_d(0x02);
            output_low(PIN_B0);
        }
        else if (x == 'c'){
            output_d(0x03);
            output_high(PIN_B1);
        }
        else if (x == 'd'){
            output_d(0x04);
            output_low(PIN_B1);
        }
        else if (x == 'e'){
            output_d(0x05);
            output_high(PIN_B2);
        }
        else if (x == 'f'){
            output_d(0x06);
            output_low(PIN_B2);
        }
    }
}
```

```

else if (x == 'g'){
    output_d(0x07);
    output_high(PIN_B3);
}
else if (x == 'h'){
    output_d(0x08);
    output_low(PIN_B3);
}
else if (x == 'i'){
    output_d(0x09);
    output_high(PIN_B4);
}

else if (x == 'j'){
    output_d(0x0A);
    output_low(PIN_B4);
}
else if (x == 'x'){
    output_d(0x00);
}
}
}

```

## APPENDIX L

### Source Code for Home Devices

File name : devices1.c

Description : This code is for the LCD, lights and fan

```
#include <p18cxxx.h>
#include "xlcd.h"
#include "delays.h"

#pragma config FOSC = HS
#pragma config WDT = OFF

//switches (input from XBee receiver circuit)
#define swlcd PORTBbits.RB2
#define swlights PORTBbits.RB1
#define swfan PORTBbits.RB0
//output
#define fan LATAbits.LATA0
#define lights LATAbits.LATA1

void Delay_1msX (unsigned int milliseconds);
void Delay_100msX (unsigned int msec);
unsigned int i, t;

void main ()
{
    //set I/O input output
    TRISA = 0b00000000;
    TRISB = 0b00001111;
    TRISD = 0b00000000;
    PORTB = 0;
    PORTD = 0;
    OpenXLCD( EIGHT_BIT & LINES_5X7 );
    ClearXLCD(); //Clear display
    //-----
    // User-defined Graphics
    //-----
    SetCGRAMAddr(0x40); //Goto CGRAM address #1
    // 90 Degree Rotated
    putcXLCD(0b11111);
    putcXLCD(0b00101);
```



```

putcXLCD(0b11101);
putcXLCD(0b00000);
putcXLCD(0b10100);
putcXLCD(0b10100);
putcXLCD(0b11111);
putcXLCD(0b00000);
// 90 Degree Rotated
putcXLCD(0b01001);
putcXLCD(0b10101);
putcXLCD(0b10011);
putcXLCD(0b01001);
putcXLCD(0b00000);
putcXLCD(0b11100);
putcXLCD(0b10100);
putcXLCD(0b11111);
// Symbol 1
putcXLCD(0b00000);
putcXLCD(0b00000);
putcXLCD(0b00000);
putcXLCD(0b00100);
putcXLCD(0b01110);
putcXLCD(0b11111);
putcXLCD(0b01110);
putcXLCD(0b01110);
// Symbol 2
putcXLCD(0b00000);
putcXLCD(0b00000);
putcXLCD(0b00100);
putcXLCD(0b01110);
putcXLCD(0b11111);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
// Symbol 3
putcXLCD(0b00000);
putcXLCD(0b00100);
putcXLCD(0b01110);
putcXLCD(0b11111);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
// Symbol 4

```

```

putcXLCD(0b00100);
putcXLCD(0b01110);
putcXLCD(0b11111);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
// Symbol 5
putcXLCD(0b01110);
putcXLCD(0b11111);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
// Symbol 6
putcXLCD(0b11111);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);
putcXLCD(0b01110);

ClearXLCD();
while(1)
{
    if(swlcd == 1){
        SetCurXLCD(0);
        putcXLCD(1);
        SetCurXLCD(20);
        putcXLCD(0);
        SetCurXLCD(2);

        for(i=2;i<8;i++){
            putcXLCD(i);
        }

        putrsXLCD(" MicroAuto");
    }
}

```

```

        } else {
            ClearXLCD();
        }

        if(swlights == 1){
            lights = 1;
        }else{
            lights = 0;
        }

        if(swfan == 1){
            fan = 1;
        }else{
            fan = 0;
        }
    }
}

void Delay_1msX (unsigned int milliseconds) {
    t=0;
    while(t<milliseconds) {
        Delay1KTCYx(11);
        Delay10TCYx(96);
        Nop();
        Nop();
        Nop();
        Nop();
        Nop();
        t++;
    }
}

void Delay_100msX (unsigned int msec) {
    t=0;
    while(t<msec) {
        Delay10KTCYx(119);
        Delay1KTCYx(9);
        Delay10TCYx(96);
        t++;
    }
}

```

File name : servo.c

Description : This code is for the servo motor in controlling the door

```
#include <pic.h>

__CONFIG ( 0x3F32 );

#define servo      RD2
#define servo2     RD0
#define SW0        RB0

#define      MHZ    *1000L
#define      KHZ    *1

#define      DelayUs(x)  { unsigned char _dcnt; \
                          _dcnt = (((x)*(20MHZ))/(24MHZ))|1; \
                          while(--_dcnt != 0) \
                              continue; \
                          _dcnt = (((x)*      (20MHZ))/(24MHZ))|1; \
                          while(--_dcnt != 0) \
                              continue; }

void DelayMs(unsigned char y);

void main(void)
{
    unsigned int i,a;

    TRISB = 0xFF;
    TRISD = 0x00;
    PORTD = 0b00000000;

    while(1){

        if(SW0 == 1){

            for(i=0;i<50;i++)
            {
                servo2=1;
                servo=1;
                DelayUs(250);
                DelayUs(250);
```

```

        DelayUs(125);

        servo=0;
        DelayMs(19);
        DelayUs(250);
        DelayUs(125);
    }

} else {

    for(i=0;i<50;i++)
    {
        servo2=0;
        servo=1;
        DelayMs(5);

        servo=0;
        DelayMs(15);
    }
}

}

void DelayMs(unsigned char y)
{
    unsigned char i;
    do {
        i = 4;
        do {
            DelayUs(250);
        } while(--i);
    } while(--y);
}

```

## APPENDIX M

### Project Gantt Charts

Note that only official academic weeks are included for both phases of Final Year Project.

#### Final Year Project 1 (January 2010 – May 2010)

Item \ Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Proposal														
Technology Evaluation														
Research on possible methods														
Development (Component 1)														

#### Final Year Project 2 (July 2010 – Nov 2010)

Item \ Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Research on possible methods														
Development (Component 1)														
Development (Component 2)														
Development (Component 3)														
Development (Component 4)														
Partial test and integration														
Full system test and integration														